Bilkent University

Department of Computer Engineering

CS353 – Database Systems

Project Design Report

Airline Company Data Management System

Website: http://cs353group20.github.io/

Group 20:

- İrem Ergün
- Nihat Eren Ekinci
- Ömer Eren
- Turan Kaan Elgin

Table of Contents

1. Revised E/R Model

Some changes were made in the E/R model, which are the following:

- There are new attributes in promotion types which are product_type, food_type and domestic respectively. Product type and food type indicate which type the promotions are valid. Domestic indicates that if the promotion is for domestic flights or not.
- There are sale_count and ticket_count attributes of ticket_staff and store_staff which show their work history.
- Seats are connected to flights rather than planes such that passengers reserve them for flights and their classes can change from flight to flight.
- There are some new attributes of plane which are model, capacity, range and altitude.
- Reservation and history relationships are revised as many to many.
- Primary key of city is changed as (city_name, country) pair such that in a country, there cannot be two cities with the same name.
- Airport is revised as a weak entity to city such that in a city, there cannot be two airports with the same name.

Figure 1.1 represents the revised E/R diagram.

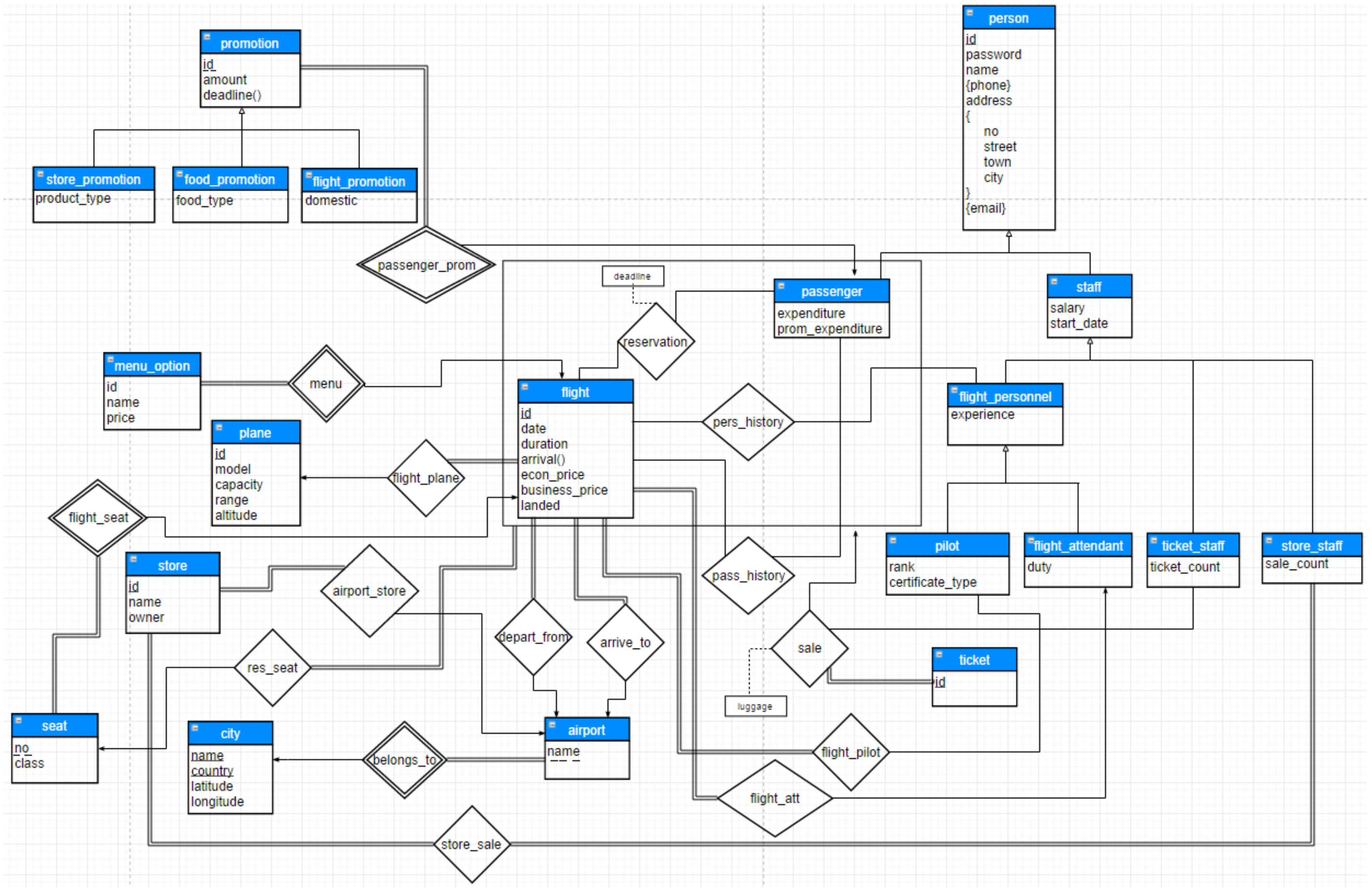


Figure 1.1 – E/R Diagram

2. Table Schemas

2.1 Person

Relational Model:
person(<u>person_id</u>, password, person_name, address_no, street, town, city)

Nontrivial Functional Dependencies:
person_id -> password person_name address_no street town city

Candidate Keys:
{(person_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE person (
        person_id int PRIMARY KEY AUTO_INCREMENT,
        password varchar(40) NOT NULL,
        person_name varchar(40) NOT NULL,
        address_no int NOT NULL,
        street varchar(40) NOT NULL,
        town varchar(40) NOT NULL);
```

2.2 PersonPhone

Relational Model:
person_phone(person_id, phone)
person_id: FK to person

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(person_id, phone)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE person_phone (
	person_id int,
	phone varchar(20),
	PRIMARY KEY(person_id, phone),
	FOREIGN KEY(person_id) referencing person);
```

2.3 PersonEmail

Relational Model:
person_email(person_id, email)
person_id: FK to person

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(person_id, email)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE person_email (
	person_id int,
	email varchar(40),
	PRIMARY KEY(person_id, email),
	FOREIGN KEY(person_id) referencing person);
```

2.4 City

Relational Model:
city(<u>city_name, country</u>, latitude, longitude)

Nontrivial Functional Dependencies:
city_name country -> latitude longitude

Candidate Keys:
{(city_name, country)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE city (
        city_name varchar(40),
        country varchar(40),
        latitude numeric(8,5) NOT NULL,
        longitude numeric(8,5) NOT NULL,
        PRIMARY KEY(city_name, country),
        check(latitude >= 0 and latitude < 360 and
                longitude >= 0 and longitude < 180));
```

2.5 Airport

Relational Model:
airport(<u>airport_name, city_name, country</u>)
(city_name, country): FK to city

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(airport_name, city_name, country)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE airport(
        airport_name varchar(40),
        city_name varchar(40),
        country varchar(40),
        PRIMARY KEY(airport_name, city_name),
        FOREIGN KEY (city_name, country) references city);
```

2.6 Store

Relational Model:
store(<u>store_id</u>, store_name, owner, airport_name, city_name, country)
(airport_name, city_name, country): FK to airport

Nontrivial Functional Dependencies:
store_id -> store_name, owner, airport_name, city_name, country

Candidate Keys:
{(store_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE store (
        store_id int PRIMARY KEY AUTO_INCREMENT,
        store_name varchar(40) NOT NULL,
        owner  varchar(40),
        airport_name varchar(40),
        city_name varchar(40),
        country varchar(40),
        FOREIGN KEY (airport_name, city_name, country) references airport);
```

2.7 Passenger

Relational Model:
passenger(pass_id, expenditure, prom_expenditure)
    pass_id: FK to person(person_id)

Nontrivial Functional Dependencies:
pass_id -> expenditure prom_expenditure

Candidate Keys:
{(pass_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE passenger (
        pass_id int PRIMARY KEY,
        expenditure numeric(4,2) NOT NULL,
        prom_expenditure numeric(4,2) NOT NULL,
        FOREIGN KEY(pass_id) references person(person_id));
```

2.8 Staff

Relational Model:
staff(staff_id, salary)
    staff_id: FK to person(person_id)

Nontrivial Functional Dependencies:
staff_id -> salary

Candidate Keys:
{(staff_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE staff (
        staff_id int PRIMARY KEY,
        salary  numeric(12,2) NOT NULL,
        FOREIGN KEY(staff_id) references person(person_id));
```

2.9 FlightPersonnel

Relational Model:
flight_personnel(<u>flight_pers_id</u>, experience)
    flight_pers_id: FK to staff(staff_id)

Nontrivial Functional Dependencies:
flight_pers_id -> experience

Candidate Keys:
{(flight_pers_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE flight_personnel (
        flight_pers_id int PRIMARY KEY,
        experience int NOT NULL,
        FOREIGN KEY(flight_pers_id) references staff(staff_id));
```

2.10 Pilot

Relational Model
pilot(<u>pilot_id</u>, rank, certificate_type)
    pilot_id: FK to flight_personnel(flight_pers_id)

Nontrivial Functional Dependencies
pilot_id -> rank certificate_type

Candidate Keys
{(pilot_id)}

Normal Form
BCNF

Table Definition
```
CREATE TABLE pilot (
        pilot_id int PRIMARY KEY,
        rank int NOT NULL,
        certificate_type enum(‘sport’, ‘recreational’, ‘private’, ‘commercial’, ‘instructor’, ‘airline
                                    transport’) NOT NULL,
        FOREIGN KEY(pilot_id) references flight_personnel(flight_pers_id));
```

2.11 FlightAttendant

Relational Model
flight_attendant(att_id, duty)
att_id: FK to flight_personnel(flight_pers_id)

Nontrivial Functional Dependencies
att_id -> duty

Candidate Keys
{(att_id)}

Normal Form
BCNF

Table Definition
```
CREATE TABLE flight_attendant (
        att_id int PRIMARY KEY,
        duty varchar(40) NOT NULL,
        FOREIGN KEY(att_id) references flight_personnel(flight_pers_id));
```

2.12 TicketStaff

Relational Model
ticket_staff(ticket_staff_id, ticket_count)
ticket_staff_id: FK to staff(staff_id)

Nontrivial Functional Dependencies
ticket_staff_id ->ticket_count

Candidate Keys
{(ticket_staff_id)}

Normal Form
BCNF

Table Definition
```
CREATE TABLE ticket_staff (
        ticket_staff_id int PRIMARY KEY,
        ticket_count int NOT NULL,
        FOREIGN KEY(ticket_staff_id) references staff(staff_id));
```

2.13 StoreStaff

Relational Model
store_staff(<u>store_staff_id</u>, sale_count, store_id)
    store_staff_id: FK to staff(staff_id)
    store-id: FK to store

Nontrivial Functional Dependencies
store_staff_id ->sale_count store_id

Candidate Keys
{(store_staff_id)}

Normal Form
BCNF

Table Definition
```
CREATE TABLE store_staff (
        store_staff_id int PRIMARY KEY,
        sale_count int NOT NULL,
        store_id int NOT NULL,
        FOREIGN KEY(store_staff_id) references staff(staff_id),
        FOREIGN KEY(store_id) references store);
```

2.14 Promotion

Relational Model

promotion(pass_id, prom_id, amount, deadline)

pass_id: FK to passenger

Nontrivial Functional Dependencies

pass_id prom_id -> amount deadline
amount -> deadline

Candidate Keys

{(pass_id, prom_id)}

Normal Form

Not 3NF (needs to be normalized)

Table Definition

```
CREATE TABLE promotion (
        pass_id int,
        prom_id int AUTO_INCREMENT,
        amount int NOT NULL,
        deadline date NOT NULL,
        PRIMARY_KEY(pass_id, prom_id),
        FOREIGN KEY(pass_id) referencing passenger);
```

2.15 StorePromotion

Relational Model
store_promotion(<u>pass_id, prom_id</u>, product_type)
    pass_id: FK to passenger
    prom_id: FK to promotion

Nontrivial Functional Dependencies
pass_id prom_id -> product_type

Candidate Keys
{(pass_id, prom_id)}

Normal Form
BCNF

Table Definition
```
CREATE TABLE store_promotion (
        pass_id int,
        prom_id int,
        product_type enum('alcohol', 'normal') NOT NULL,
        PRIMARY_KEY(pass_id, prom_id),
        FOREIGN KEY(pass_id) referencing passenger
        FOREIGN KEY(prom_id) referencing promotion);
```

2.16 FoodPromotion

Relational Model
food_promotion(pass_id, prom_id, food_type)
pass_id: FK to passenger
prom_id: FK to promotion

Nontrivial Functional Dependencies
pass_id prom_id -> food_type

Candidate Keys
{(pass_id, prom_id)}

Normal Form
BCNF

Table Definition
```
CREATE TABLE food_promotion (
	pass_id int,
	prom_id int,
	food_type enum('meal', 'drink') NOT NULL,
	PRIMARY_KEY(pass_id, prom_id),
	FOREIGN KEY(pass_id) referencing passenger
	FOREIGN KEY(prom_id) referencing promotion);
```

2.17 FlightPromotion

Relational Model:
flight_promotion(<u>pass_id, prom_id</u>, domestic)
    pass_id: FK to passenger
    prom_id: FK to promotion

Nontrivial Functional Dependencies:
pass_id, prom_id -> domestic

Candidate Keys:
{(pass_id, prom_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE flight_promotion (
        pass_id int,
        prom_id int,
        domestic binary NOT NULL,
        PRIMARY KEY(pass_id, prom_id),
        FOREIGN KEY(pass_id) referencing passenger
        FOREIGN KEY(prom_id) referencing promotion);
```

2.18 Plane

Relational Model:
plane(plane_id, model, capacity, range, altitude)

Functional Dependencies:
plane_id -> model
model -> capacity range altitude

Candidate Keys:
{(plane_id)}

Normal Form:
Not 3NF (needs to be normalized)

Table Definition:
```
CREATE TABLE plane(
        plane_id int PRIMARY KEY AUTO_INCREMENT,
        model varchar(20) NOT NULL,
        capacity int NOT NULL,
        range numeric(5,2) NOT NULL,
        altitude numeric(5,2) NOT NULL);
```

2.19 Flight

Relational Model:
flight(flight_id, date, plane_id, dep_airport_name, dep_city_name, dep_country, arr_airport_name, arr_city_name, arr_country, duration, arrival, econ_price, business_price, landed)
plane_id: FK to plane
(dep_airport_name, dep_city_name, dep_country): FK to airport(airport_name, city_name, country)
(arr_airport_name, arr _city_name, arr _country): FK to airport(airport_name, city_name, country)

Functional Dependencies:
plane_id -> date, plane_id, dep_airport_name, dep_city_name, dep_country, arr_airport_name, arr_city_name, arr_country, duration, arrival, econ_price, business_price, landed
duration -> arrival

Candidate Keys:
{(plane_id)}

Normal Form:
Not 3NF (needs to be normalized)

Table Definition:
```
CREATE TABLE flight (
	flight_id int PRIMARY KEY AUTO_INCREMENT,
	date DATETIME NOT NULL,
	plane_id int,
	dep_airport_name varchar(40,
	dep_city_name varchar(40,
	dep_country varchar(40),
	arr_airport_name varchar(40),
	arr _city_name varchar(40),
	arr _country varchar(40),
	duration int NOT NULL,
	arrival int NOT NULL,
	econ_price numeric(6,2),
	business_price numeric(6,2),
	landed binary NOT NULL,
	FOREIGN KEY(plane_id) references plane
	FOREIGN KEY(dep_airport_name, dep_city_name, dep_country) references
	airport(airport_name, city_name, country)
	FOREIGN KEY(arr_airport_id, arr_city_name, arr_country) references
	airport(airport_name, city_name, country));
```

2.20 Seat

Relational Model:
seat(<u>flight_id, no</u>, class)
flight_id: FK to flight

Functional Dependencies:
flight_id no -> class

Candidate Keys:
{(flight_id, no)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE seat (
        flight_id int,
        no int,
        class enum('econ', 'business') NOT NULL,
        PRIMARY KEY (flight_id, no),
        FOREIGN KEY (plane_id) references plane);
```

2.21 MenuOption

Relational Model:
menu_option(<u>flight_id, option_id</u>, option_name, price)
flight_id: FK to flight

Functional Dependencies:
flight_id option_id -> option_name price

Candidate Keys:
{(flight_id, option_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE menu_option(
        flight_id int,
        option_id int,
        option_name varchar(40) NOT NULL,
        PRIMARY KEY (flight_id, option_id),
        FOREIGN KEY (flight_id) references flight);
```

2.22 Reservation

Relational Model:
reservation(<u>flight_id, pass_id</u>, deadline, seat_no)
    flight_id: FK to flight
    pass_id: FK to passenger
    seat_no: FK to seat

Functional Dependencies:
(flight_id, pass_id) -> deadline seat_no

Candidate Keys:
{(flight_id, pass_id)}

Normal Form:
```
CREATE TABLE reservation(
        flight_id int,
        pass_id int,
        deadline date NOT NULL,
        seat_no int NOT NULL,
        PRIMARY KEY(flight_id, pass_id),
        FOREIGN KEY(flight_id) references flight,
        FOREIGN KEY(pass_id) references passenger
        FOREIGN KEY(seat_no) references seat);
```

2.23 PassengerHistory

Relational Model:
pass_history(flight_id, pass_id)
    flight_id: FK to flight
    pass_id: FK to passenger

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(flight_id, pass_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE pass_history (
        flight_id int,
        pass_id int,
        PRIMARY KEY(flight_id, pass_id),
        FOREIGN KEY(flight_id) references flight,
        FOREIGN KEY(pass_id) references passenger);
```

2.24 PersonnelHistory

Relational Model:
pers_history(<u>flight_id, flight_pers_id</u>)
flight_id: FK to flight
flight_pers_id: FK to flight_personnel

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(flight_id, flight_pers_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE pers_history(
        flight_id int,
        pers_id int,
        PRIMARY KEY(flight_id, flight_pers_id),
        FOREIGN KEY(flight_id) references flight,
        FOREIGN KEY(flight_pers_id) references flight_personnel);
```

2.25 Flight-Pilot Relationship

Relational Model:
flight_pilot(<u>flight_id, pilot_id</u>)
    flight_id: FK to flight
    pilot_id: FK to pilot

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(flight_id, pilot_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE flight_pilot (
        flight_id int,
        pilot_id int,
        PRIMARY KEY(flight_id, pilot_id),
        FOREIGN KEY(flight_id) references flight,
        FOREIGN KEY(pilot_id) references pilot);
```

2.26 Flight – FlightAttendant Relationship

Relational Model:
flight_att(<u>flight_id, att_id</u>)
    flight_id: FK to flight
    att_id: FK to flight_attendant

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(flight_id, att_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE flight_att(
        flight_id int,
        att_id int,
        PRIMARY KEY(flight_id, att_id),
        FOREIGN KEY(flight_id) references flight,
        FOREIGN KEY(att_id) references flight_attendant);
```

2.27 Ticket

Relational Model:
ticket(<u>ticket_id</u>, flight_id, pass_id, staff_id, luggage)

Nontrivial Functional Dependencies:
ticket_id -> price flight_id pass_id staff_id luggage

Candidate Keys:
{(ticket_id)}

Table Definition:
```
CREATE TABLE ticket(
        ticket_id int,
        flight_id int NOT NULL,
        pass_id int NOT NULL,
        staff_id int,
        luggage int NOT NULL,
        PRIMARY KEY(ticket_id),
        FOREIGN KEY(flight_id, pass_id) references reservation,
        FOREIGN KEY(staff_id) references ticket_staff);
```

3. Functional Dependencies and Normalization of Tables

In the E/R model, there is an entity called *plane* which has the following table schema:

plane(<u>id</u>, model, capacity, range, altitude)

Each model of a plane has unique capacity, range and altitude values, so there is a functional dependency which is model -> capacity range altitude. So, the table should be normalized by decomposing it into *plane* and *plane_model* tables which are the following:

Plane

Relational Model:
plane(<u>plane_id</u>, model)

Functional Dependencies:
plane_id -> model

Candidate Keys:
{(plane_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE plane(
        plane_id int PRIMARY KEY,
        model varchar(20) NOT NULL,
        FOREIGN KEY (model) references plane_model);
```

PlaneModel

Relational Model:
plane_model(model, capacity, range, altitude)
    model: FK to plane

Functional Dependencies:
model -> capacity range altitude

Candidate Keys:
{(model)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE plane_model(
        model varchar(20) PRIMARY KEY,
        capacity int NOT NULL,
        range numeric(5,2) NOT NULL,
        altitude numeric(5,2) NOT NULL,
        FOREIGN KEY(model) referencing plane);
```

Another entity which needs to be normalized is *promotion* since deadline is a derived attribute which can be determined by knowing its amount and the current date since there deadlines are fixed and mapped to amounts. So, the table should be normalized by decomposing it into *promotion* and *promotion_deadline* tables which are the following:

Promotion

Relational Model
promotion(pass_id, prom_id, amount)
    pass_id: FK to passenger

Nontrivial Functional Dependencies
pass_id prom_id -> amount deadline

Candidate Keys
{(pass_id, prom_id)}

Normal Form
BCNF

Table Definition
```
CREATE TABLE promotion (
        pass_id int,
        prom_id int AUTO_INCREMENT,
        amount int NOT NULL,
        PRIMARY_KEY(pass_id, prom_id),
        FOREIGN KEY(pass_id) referencing passenger);
```

PromotionDeadline

Relational Model:
promotion_deadline(amount, deadline)
    amount: FK to promotion

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(amount, deadline)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE promotion_deadline(
        amount int,
        deadline date,
        PRIMARY KEY(amount, deadline),
        FOREIGN KEY(amount) referencing promotion);
```

The entity *flight* also needs to be normalized such that *arrival* is a derived attribute which can be determined by *date* and *duration* attributes.

Flight

Relational Model:
flight(<u>flight_id</u>, date, plane_id, dep_airport_id, arr_airport_id, duration, econ_price, business_price)

plane_id: FK to plane
dep_airport_id: FK to airport(airport_id)
arr_airport_id: FK to airport(airport_id)

Functional Dependencies:
flight_id -> date plane_id dep_airport_id arr_airport_id duration econ_price business_price

Candidate Keys:
{(flight_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE flight (
	flight_id int PRIMARY KEY AUTO_INCREMENT,
	date DATE NOT NULL,
	plane_id int,
	dep_airport_id int,
	arr_airport_id int,
	duration numeric(3,2) NOT NULL,
	econ_price numeric(6,2),
	business_price numeric(6,2),
	FOREIGN KEY (plane_id) references plane,
	FOREIGN KEY(dep_airport_id) references airport(airport_id),
	FOREIGN KEY(arr_airport_id) references airport(airport_id));
```

FlightArrival

Relational Model:
flight_arrival(<u>date, duration, arrival</u>)
(date, duration): FK to flight

Nontrivial Functional Dependencies:
None

Candidate Keys:
{(date, duration, arrival)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE flight_arrival (
        date DATE,
        duration numeric(3,2),
        arrival DATE,
        PRIMARY KEY(date, duration, arrival),
        FOREIGN KEY(date, duration) referencing flight);
```

4. Functional Components

4.1 Use Cases / Scenarios

In the system, there are six types of users which are passengers, pilots, flight attendants, ticket staff, store staff, and admin. All users should login to the system by their registered accounts when using the system. The functionalities of the system differ for different types of users.

Passenger:

- Passenger will be able to view his/her profile.
- Passenger will be able to change his/her password.
- Passenger will be able to browse flights.
- Passenger will be able to make reservations for flights and choose their seats.
- Passenger will be able to cancel their reservations.
- Passenger will be able to purchase tickets either with or without reservation.
- Passenger will be able to cancel their tickets by paying some penalty.
- Passenger will be able to change their seats after reservation.
- Passenger will be able to make reservations for connecting flights.
- Passenger will be able to see their flight histories.
- Passenger will be able to see food menus of flights and stores of airports.
- Passenger will be able to buy extra luggage.
- System will be able to drop reservations after a deadline for purchasing tickets.
- System will be able to give promotions to passengers according to purchasing tickets.

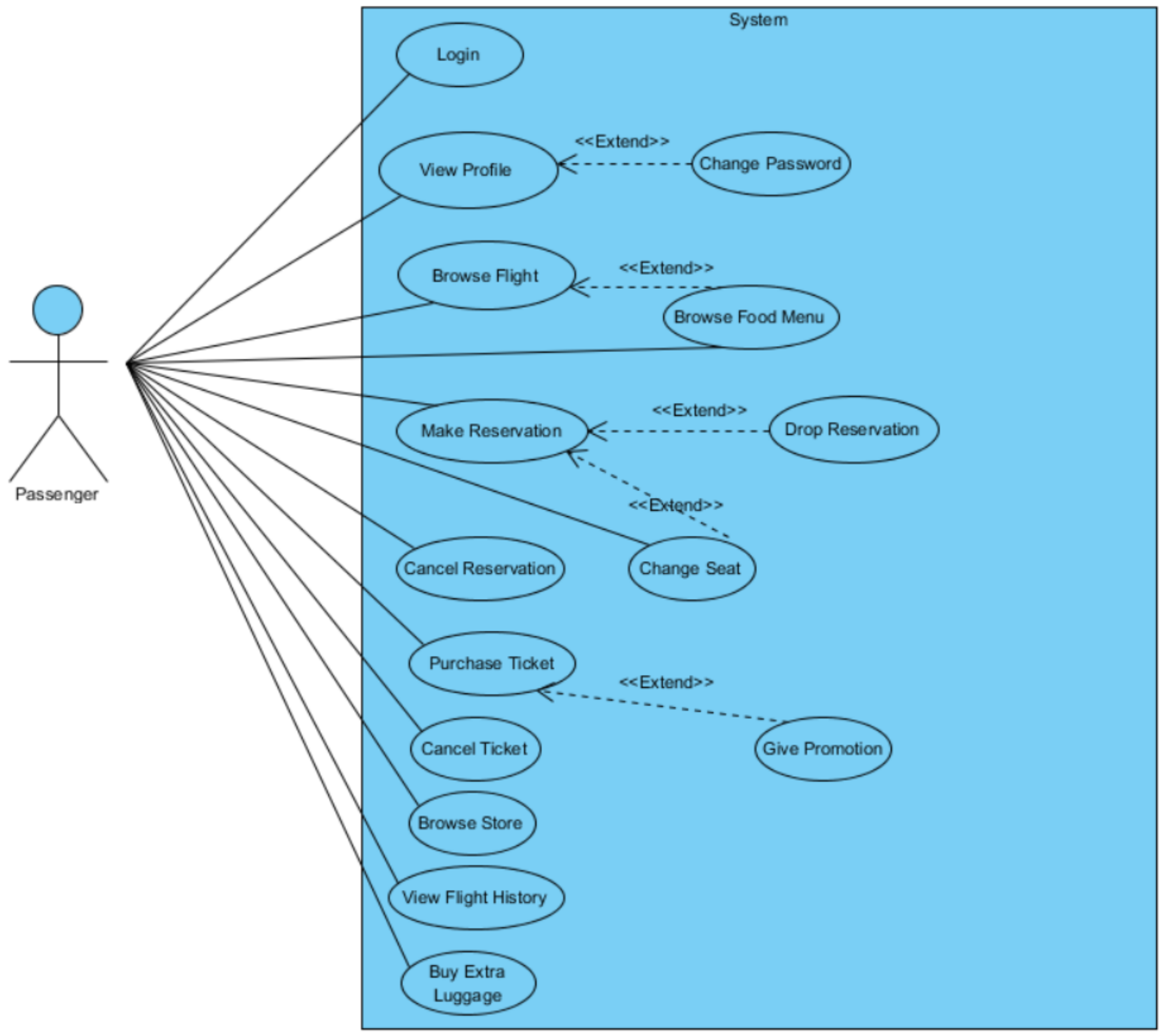


Figure 4.1 – Use Case Diagram of Passenger

Flight Personnel:

- Flight personnel will be able to view their scheduled flights.
- Flight personnel will be able to view their flight history.
- Flight attendant will be able to record usage of food promotions of the passengers to the system.
- Pilot will be able to browse information about the planes of their flights.

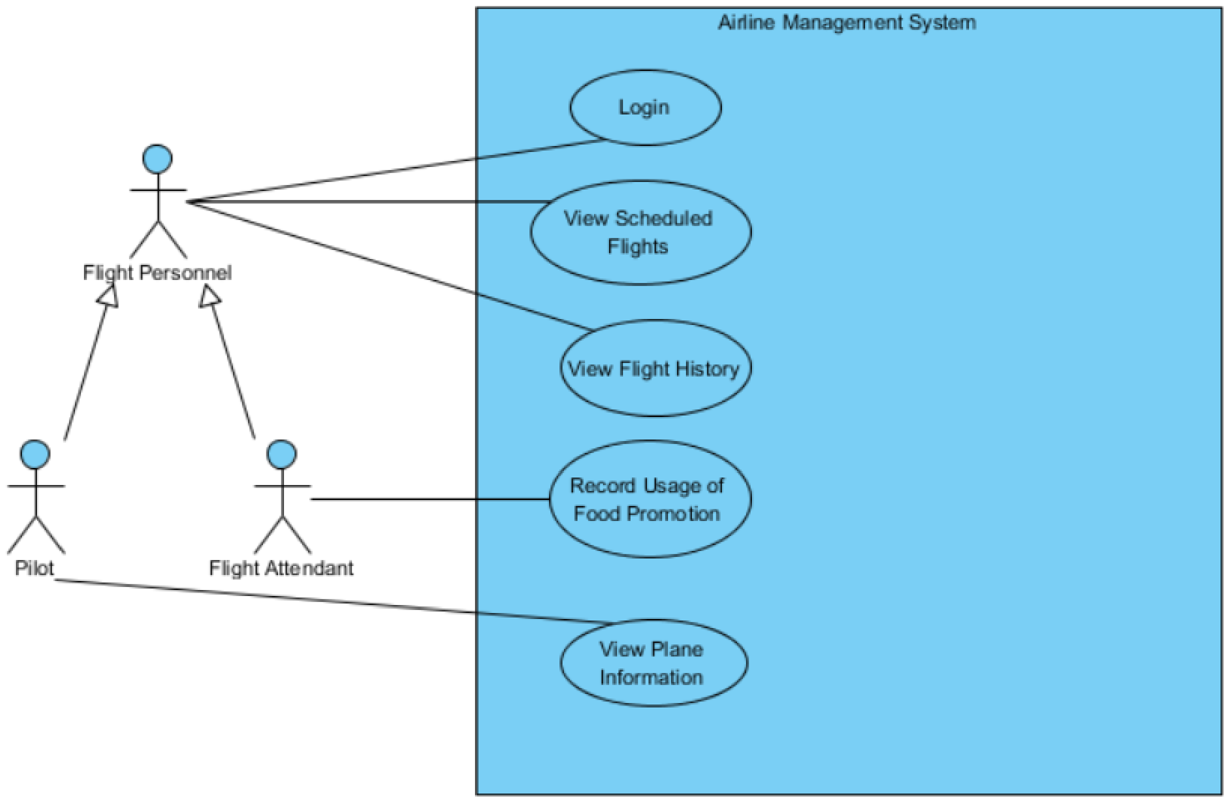


Figure 4.2 - Use Case Diagram of Flight Personnel

Company Staff:

- Ticket staff will be able to record the ticket sales to the system.
- Ticket staff will be able to make and cancel reservations.
- Store staff will be able to record usage of store promotions of the passengers to the system.
- System will be able to give promotions to passenegers according to the ticket sales.

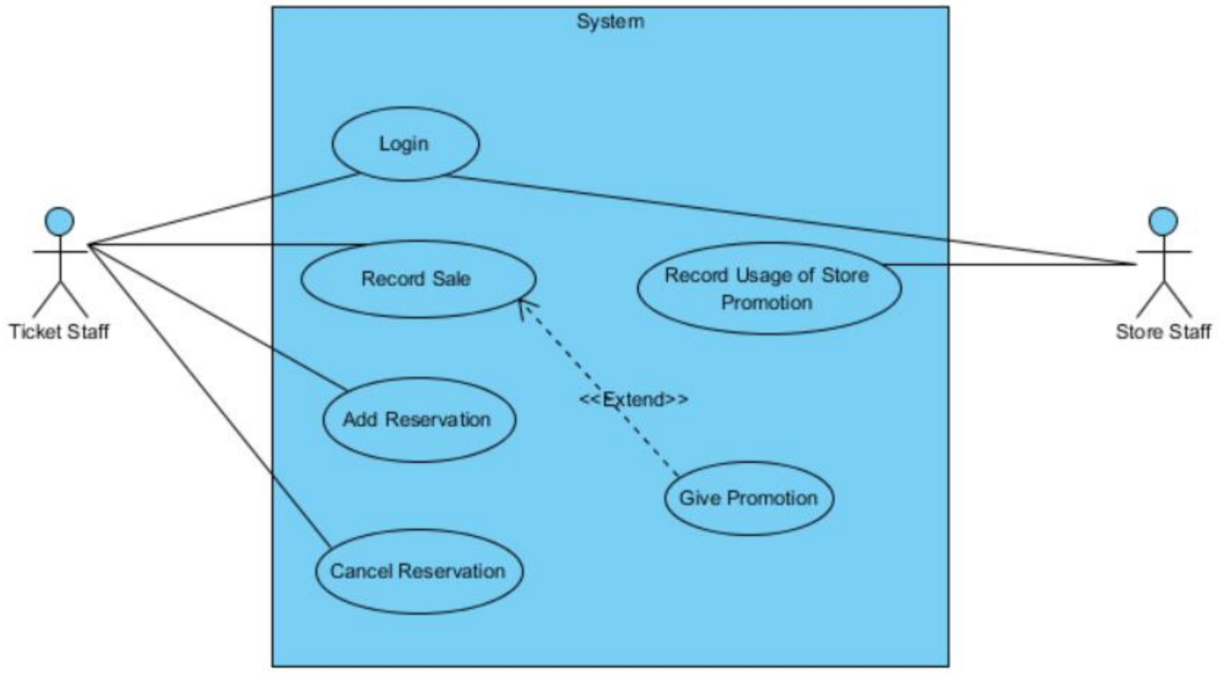


Figure 4.3 - Use Case Diagram of Company Staff

Admin:

- Admin will be able to assign users to particular duties such as pilot, flight attendant, and so on.
- Admin will be able to remove users from the system.
- Admin will be able to add flights to the system.
- Admin will be able to remove flights from the system.
- Admin will be able to delay flights.
- Admin will be able to add food menu options to the system.
- Admin will be able to remove food menu options from the system.
- Admin will be able to add stores to the system.
- Admin will be able to remove stores from the system.
- Admin will be able to mark flights as landed.

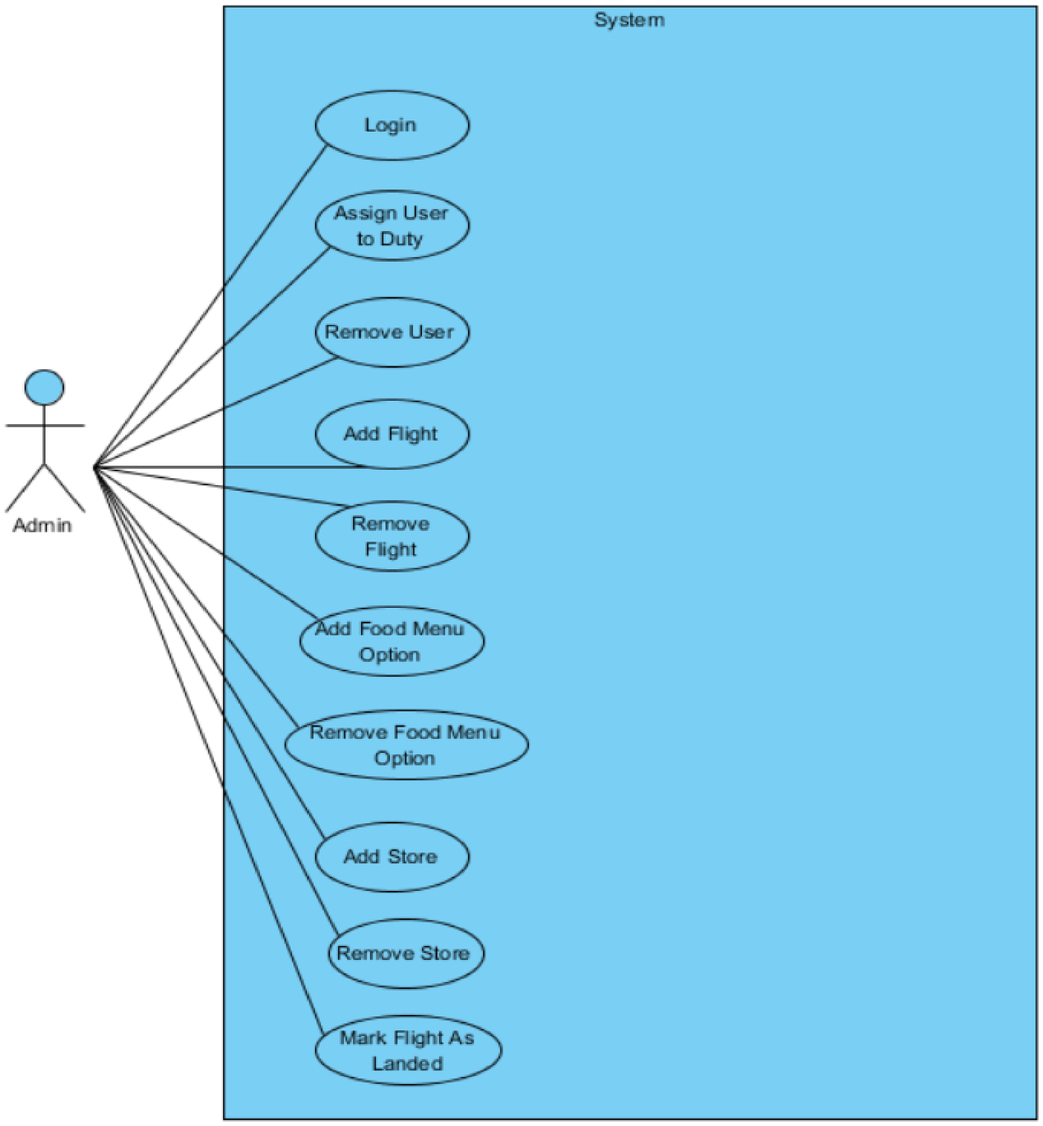


Figure 4.4 - Use Case Diagram of Admin

4.2 Algorithms

4.2.1 Ticket-Related Algorithms

Purchasing a Ticket

In the system, each ticket is associated with a reservation between a passenger and a flight. However, the system also supports purchasing tickets before making reservations. In that case, reservations are done automatically by the system.

Canceling a Ticket

While canceling a ticket, at first the system checks if the deadline is passed or not. If it is not passed; at first, it removes the particular reservation from the system, then it removes the ticket.

4.2.2 Promotion-Related Algorithms

Giving Promotions

When passengers purchase a ticket or a ticket sale is recorded by a ticket staff, the particular flight and passenger will be added to the *pass_history* relationship. Also, price of the ticket will be added to the *expenditure* attribute of passenger. Then, the system compares the expenditure with some particular values which are the values of promotion limits. At first, it checks whether there is a flight promotion to be given. If there is, it adds the promotion to the database, and reduces the value of expenditure by the value of promotion. It repeats this process until there is not any more flight promotions. Then it repeats the process for store and food promotions respectively.

Using Promotions

Company staff enters usage of promotions. After getting ID of the passenger, the system displays the promotions of the passenger and deletes the selected one from the database.

4.2.3 Reservation-Related Algorithms

Making Reservations

The system displays the available cities and according to the selected start and destination cities, it displays all of the available flights including connecting ones. For that purpose, it needs to keep a directed graph with cities as nodes and flights as connections. It will traverse the graph starting from the start city and ending at the destination city, and displays all available paths using a depth first search algorithm. According to the selected flight, the system displays the seats of the flights. Then it sets the *reserved* attribute of the selected seats.

Delaying a Connecting Flight

When a flight is delayed, the system checks if it is included in a connecting flight or not. If it is, then it will reshape the city-flight graph and assigns the previous reservations to the closest flight. Then, it will add a flight promotion for the particular passengers.

4.2.4 Admin-Related Algorithms

Delaying Flights

When admin delays a flight, the *date* attribute of the *flight* table will be updated. Since *arrival* is a derived attribute, it will be updated automatically by using a trigger. Then, the algorithm in the previous section will be computed for changing reservations of connecting flights.

Removing Flights

When admin remove a flight, the reservations will be removed first. Then, the reservation will be automatically assigned to the closest flight with the particular start and destination cities and schedules of flight personnel will be updated accordingly.

Removing Users

When admin removes a user from the system, the records of the particular user such as flight history, reservations and promotions will be removed firstly.

4.2.5 Addressing Logical Requirements with Algorithms

For eliminating logical errors, the following cases should be considered.

Arrival Times of Flights

The arrival times of flights is assigned to the sum of date and duration attributes.

Deadlines of Promotions

The start dates of the promotions will be the current date. The deadlines of the promotions should be at least the sum of current date and the minimum value of their durations.

4.3 Data Structures

For the attribute domains; int, numeric, varchar, date, datetime and binary types of SQL will be used. For connecting flights, a directed graph will be used in which nodes are the cities and connections are the flights. The graph may be disconnected such that there may be no flights for a particular city at that time.

5 User Interface Design and Corresponding SQL Statements

5.1 Login Screen

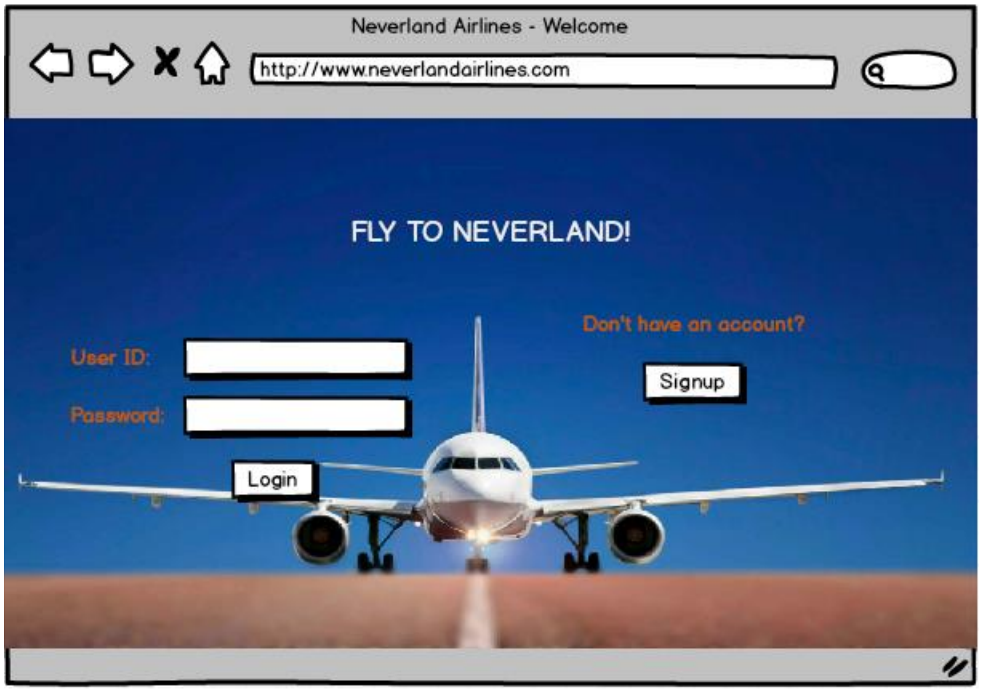


Figure 5.1 – Login Screen

Inputs: @person_id, @password

Process: By the initial login screen, all users login to the system by entering their ID numbers and passwords. Also, they can create their accounts by clicking on the sign up button.

SQL Statements:

Finding the Particular Person

```
SELECT *
FROM person
WHERE person_id = @person_id AND password = @password
```

5.2 Create Account Screen

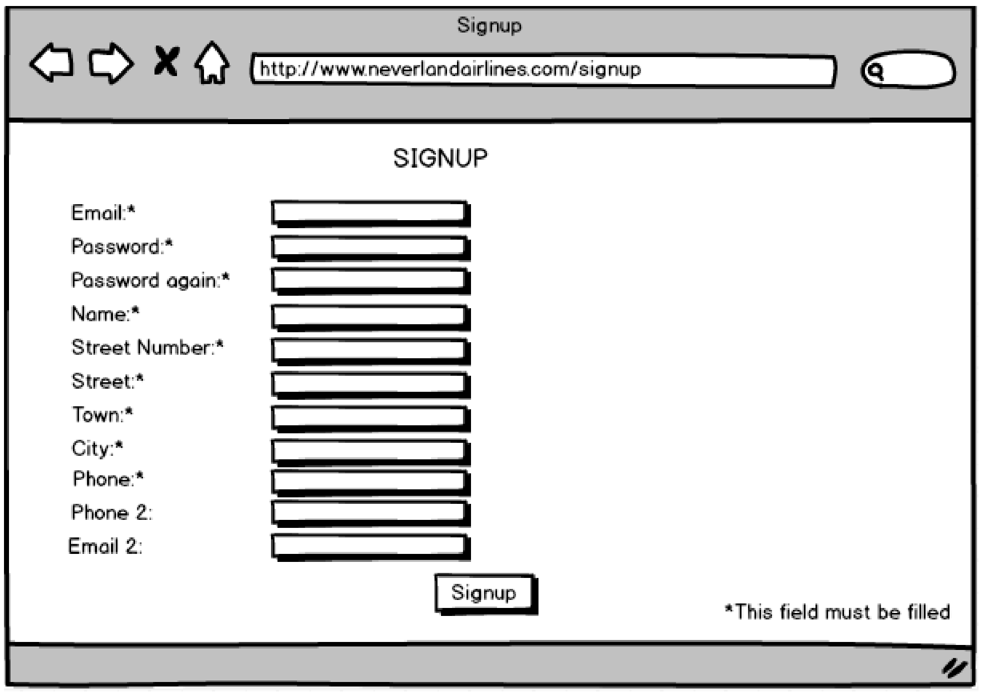


Figure 5.2 – Create Account Screen

Inputs: @person_id, @password, @person_name, @street_no, @street, @town, @city, @phone, @email

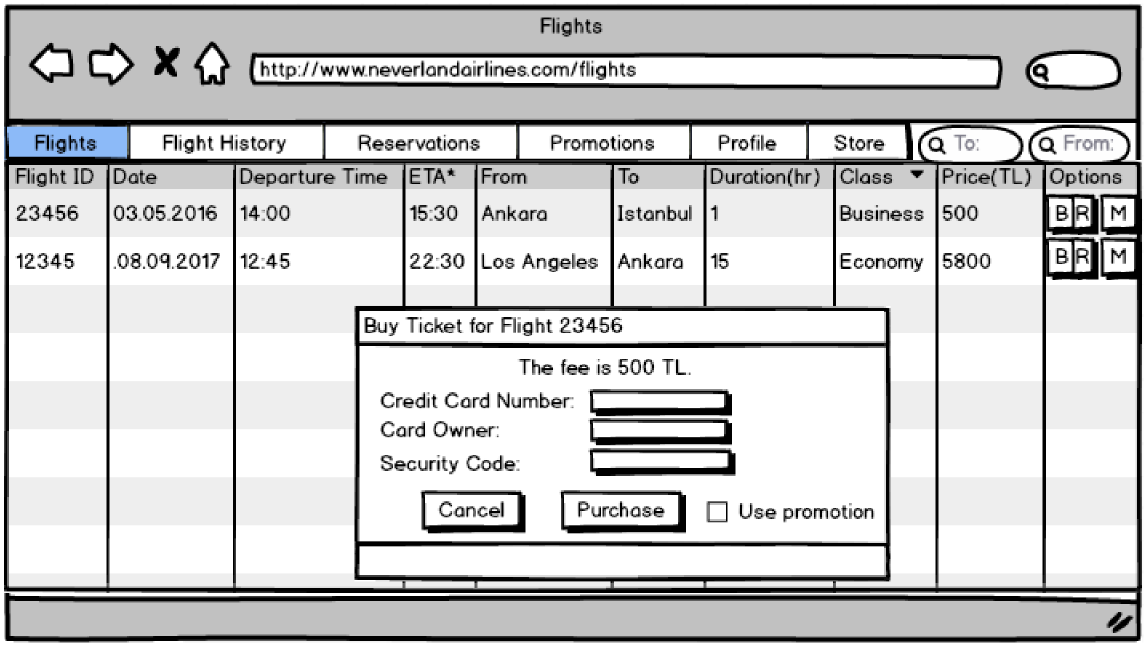
Process: The create account screen is used for registering to the system. Users can create their accounts by entering the necessary information. At first, all users can register as passengers. Then, if they have different duties, system administrator can assign them to those duties.

SQL Statements:

Creating an Account

```
INSERT INTO person(person_id, person_name, password, no, street, town, city)
VALUES(@person_id, @person_name, @password, @add_no, @street, @town, @city)

INSERT INTO passenger(pass_id, luggage, expenditure)
VALUES(@person_id, 10, 0)

INSERT INTO person_phone(person_id, phone)
VALUES(@person_id, @phone)

INSERT INTO person_email(person_id, email)
VALUES(@person_id, @email)
```

5.3 Passenger Flights Screen



Flight ID	Date	Departure Time	ETA*	From	To	Duration(hr)	Class	Price(TL)	Options
23456	03.05.2016	14:00	15:30	Ankara	Istanbul	1	Business	500	B R M
12345	08.09.2017	12:45	22:30	Los Angeles	Ankara	15	Economy	5800	B R M

Buy Ticket for Flight 23456

The fee is 500 TL.

Credit Card Number:
Card Owner:
Security Code:

Cancel Purchase ☐ Use promotion

Figure 5.3 – Passenger Flights Screen

Inputs: @pass_id, @source_filter, @dest_filter, @flight_id, @staff_id

@staff_id: ID of the ticket staff who record sale of the ticket (If the ticket is purchased online, it will be null)

Process: In the passenger screen, there are six options. Passengers can filter the list of flights by providing source and destination cities to the particular text fields and pressing the enter button. Secondly, they can view their flight histories. Thirdly, they can view their reservations. Also, they can view their promotions and profile. In addition to those, they can view stores of airports.

In the flights option, after filtering flights, passengers are provided three options which are buy (B button), reserve (R button), view menu options (M button). They can also select their classes as business and economy. When they click on the buy option, a popup window appears in which they provide their credit card information. They also choose to use promotion.

SQL Queries:

Filtering the List of Flights

```
SELECT *
FROM flight F NATURAL JOIN flight_arrival
WHERE (SELECT city_name, country
            FROM airport A
            WHERE F.dep_airport_name = A.airport_id) LIKE '%@source_filter%'
          AND (SELECT city_id
                  FROM airport A
                  WHERE F.arr_airport_id = A.airport_id) LIKE '%@dest_filter%'
ORDER BY F.flight_id
```

Making Reservations

```
INSERT INTO reservation(pass_id, flight_id, deadline)
VALUES(@pass_id, @flight_id, @deadline)
```

Note: @deadline is provided by the system.

Purchasing Ticket

```
INSERT INTO ticket(ticket_id, flight_id, pass_id, staff_id)
VALUES(@ticket_id, @flight_id, @pass_id, @staff_id)
```

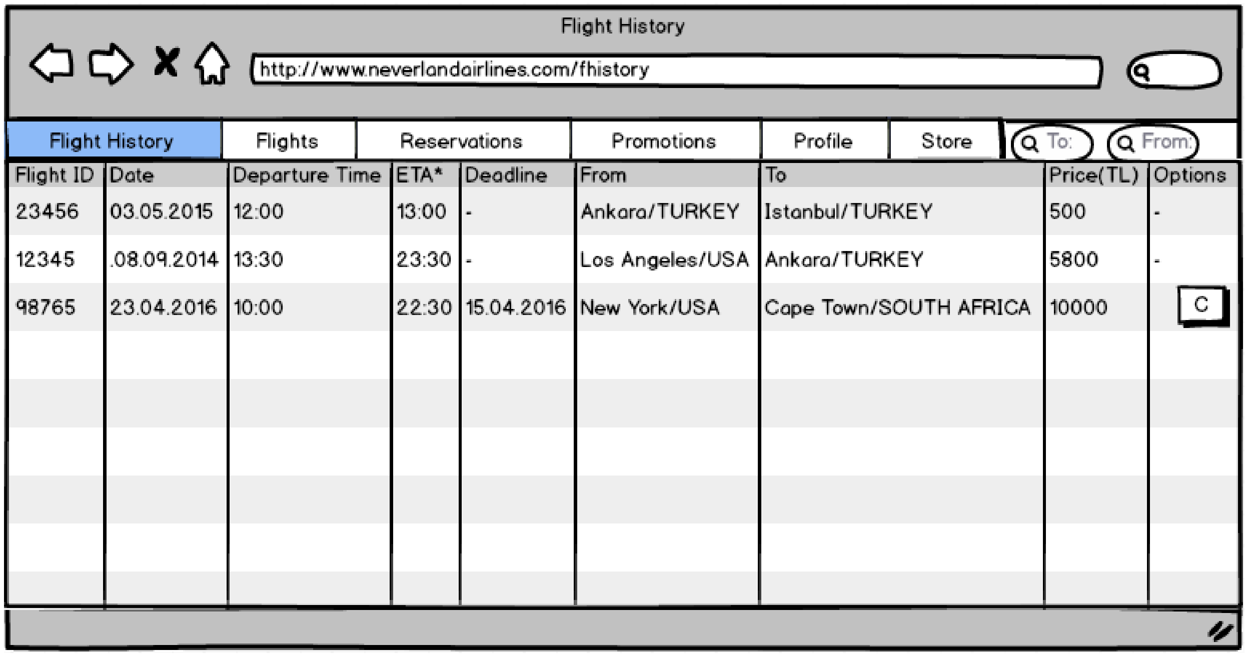
Note: @ticket_id is provided by the system.

Then, expenditure of the particular passenger will be increased by a trigger.

Canceling

No query needed.

5.4 Passenger Flight History Screen



Flight ID	Date	Departure Time	ETA*	Deadline	From	To	Price(TL)	Options
23456	03.05.2015	12:00	13:00	-	Ankara/TURKEY	Istanbul/TURKEY	500	-
12345	.08.09.2014	13:30	23:30	-	Los Angeles/USA	Ankara/TURKEY	5800	-
98765	23.04.2016	10:00	22:30	15.04.2016	New York/USA	Cape Town/SOUTH AFRICA	10000	C

Figure 5.4 – Passenger Flight History Screen

Inputs: @pass_id, @flight_id

@staff_id: ticket staff's id who cancel the ticket

Process: In the flight history screen, passengers can view both their previous flights and currently purchased flights. They can cancel their tickets for current flights by the C button.

SQL Queries:

Displaying Flight History

```
SELECT *
FROM pass_history_view
```

In this query, pass_history_view will be used which is created in Section 6.1.

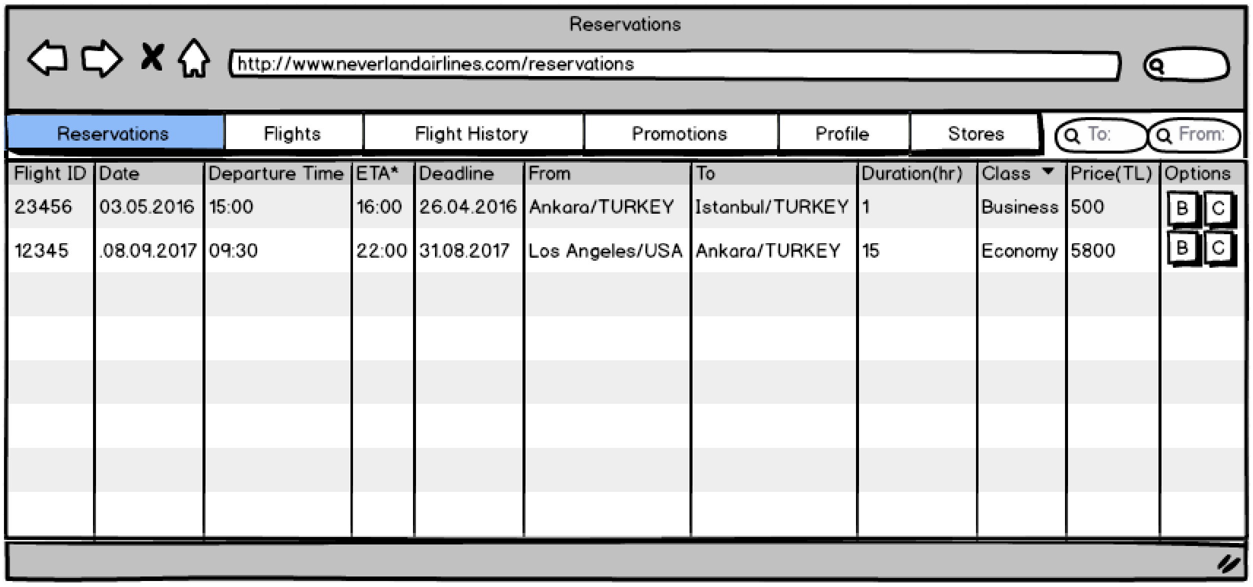
Canceling Tickets

```
DELETE FROM ticket
WHERE pass_id = @pass_id AND flight_id = @flight_id
```

Then, expenditure of the particular passenger will be decreased by a trigger.

Canceling
No query needed.

5.5 Passenger Reservations Screen



Flight ID	Date	Departure Time	ETA*	Deadline	From	To	Duration(hr)	Class	Price(TL)	Options
23456	03.05.2016	15:00	16:00	26.04.2016	Ankara/TURKEY	Istanbul/TURKEY	1	Business	500	B C
12345	.08.09.2017	09:30	22:00	31.08.2017	Los Angeles/USA	Ankara/TURKEY	15	Economy	5800	B C

Figure 5.5 – Passenger Reservations Screen

Inputs: @pass_id, @flight_id

Process: In the reservations screen, passengers can see their current reservations. They can buy their tickets by clicking on the B button and cancel them by clicking on C button.

SQL Queries:

Displaying Reservations

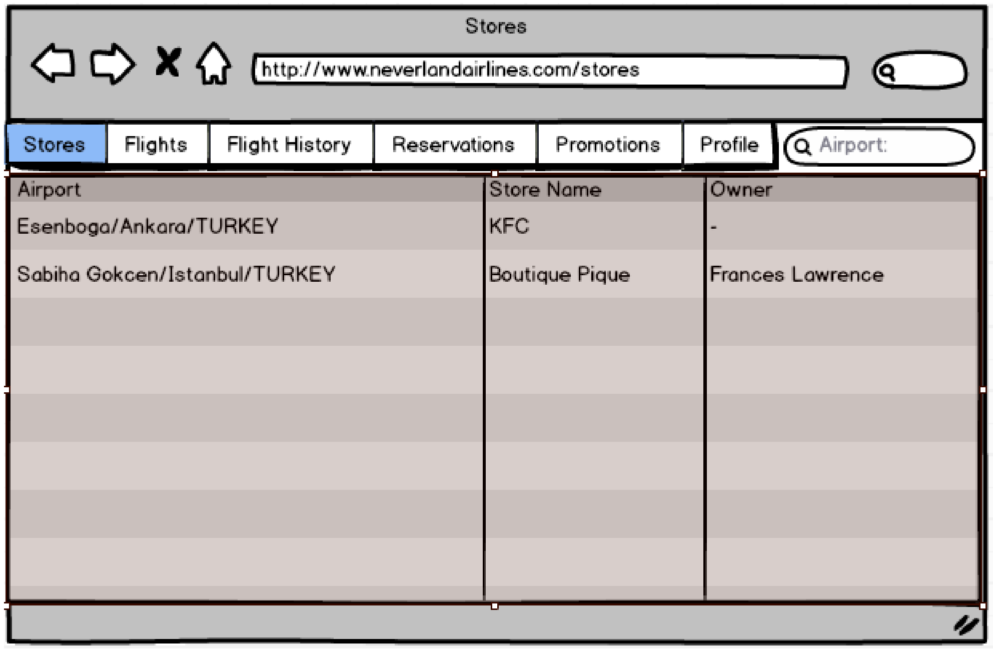
```
SELECT *
FROM reservation
WHERE pass_id = @pass_id
```

Canceling Reservations

```
DELETE FROM reservation
WHERE pass_id = @pass_id AND flight_id = @flight_id
```

Canceling
No query needed.

5.6 Passenger Store View Screen



Airport	Store Name	Owner
Esenboga/Ankara/TURKEY	KFC	-
Sabiha Gokcen/Istanbul/TURKEY	Boutique Pique	Frances Lawrence

Figure 5.6 – Passenger Store View Screen

Inputs: @airport_filter

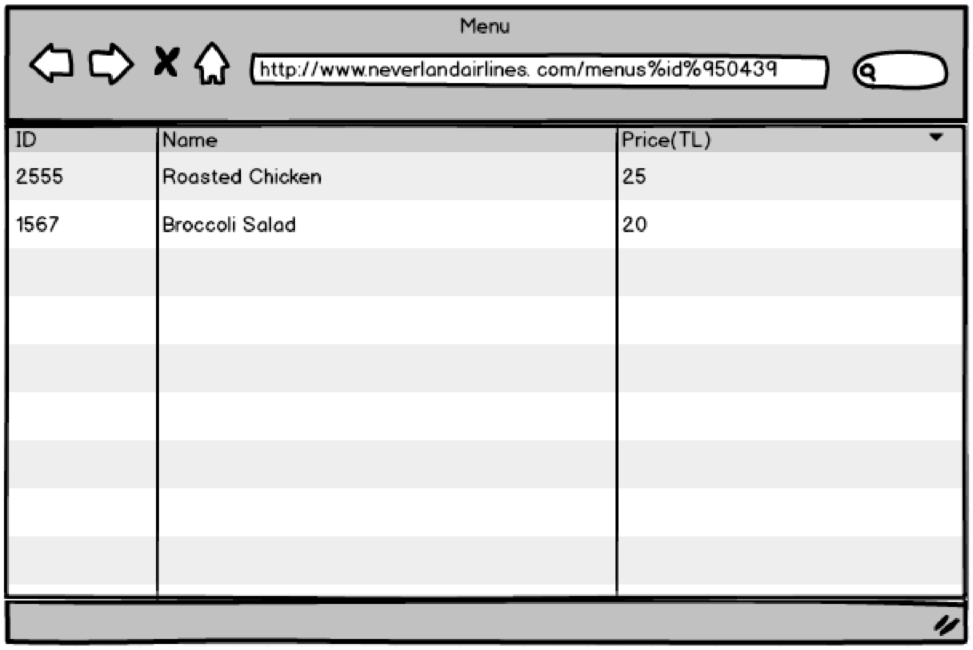
Process: In the store view screen, passengers can browse stores of airports by filtering them by their names.

Browsing Stores

SELECT store_id, store_name, owner, airport_name, city_name, country
FROM store NATURAL JOIN airport
WHERE airport_name LIKE '%@airport_filter%'

Canceling
No query needed.

5.7 Passenger Menu Option View Screen



ID	Name	Price(TL)
2555	Roasted Chicken	25
1567	Broccoli Salad	20

Figure 5.7 – Passenger Menu Option View Screen

Inputs: @flight_id

Process: In the menu option view screen, passengers can see the menu options of a particular flight, by clicking on the M button in the flights screen. They can change the money unit by selecting it from the price column.

SQL Queries:

Browsing Menu Options

```
SELECT option_id, option_name, price
FROM menu_option
WHERE flight_id = @flight_id
```

Canceling
No query needed.

5.8 Profile Screen

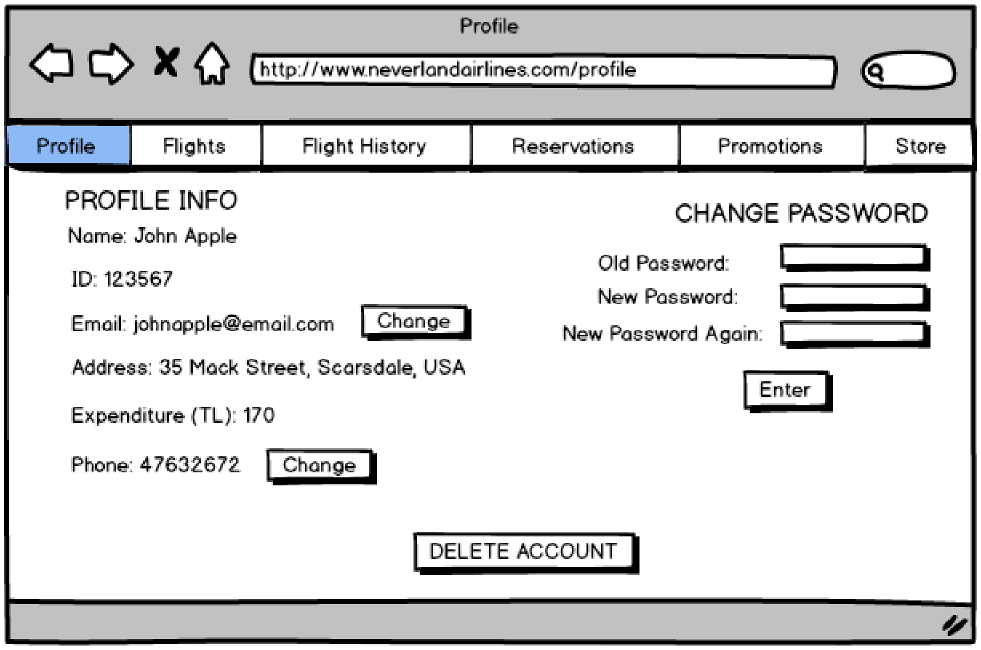


Figure 5.8 – Profile Screen

Inputs: @person_id, @old_password, @new_password, @phone, @email

Process: All users are able to see their profile information in profile screen. They can also change their passwords, emails and phone numbers and delete their accounts by this screen.

SQL Statements:

Displaying User Information

```
SELECT person_id, person_name, salary, street_no, street, town, city, luggage, expenditure
FROM person
WHERE person_id IN (SELECT pass_id
                    FROM passenger)
        AND person_id = @person_id

SELECT person_id, person_name, salary, street_no, street, town, city, salary, start_date,
experience, rank, certificate_type
FROM person
WHERE person_id IN (SELECT pilot_id
                    FROM pilot)
        AND person_id = @person_id
```

```
SELECT person_id, person_name, salary, street_no, street, town, city, salary, start_date,
experience, duty
FROM person
WHERE person_id IN (SELECT att_id
                    FROM flight_attendant)
        AND person_id = @person_id

SELECT person_id, person_name, salary, street_no, street, town, city, ticket_count
FROM person
WHERE person_id IN (SELECT ticket_staff_id
                    FROM ticket_staff)
        AND person_id = @person_id

SELECT person_id, person_name, salary, street_no, street, town, city, sale_count
FROM person
WHERE person_id IN (SELECT store_staff_id
                    FROM store_staff)
        AND person_id = @person_id

WITH pass_id_table(pass_id) AS
(SELECT pass_id
FROM passenger),
pilot_id_table(pilot_id) AS
(SELECT pilot_id
FROM pilot),
att_id_table(att_id) AS
(SELECT att_id
FROM flight_attendant),
store_st_id_table(store_staff_id) AS
(SELECT store_staff_id
FROM store_staff),
ticket_st_id_table(ticket_staff_id) AS
(SELECT ticket_staff_id
FROM ticket_staff)
SELECT CASE
             WHEN person_id IN pass_id_table THEN ‘Passenger’
             WHEN person_id IN pilot_id_table THEN ‘Pilot’
             WHEN person_id IN att_id_table THEN ‘Flight Attendant’
             WHEN person_id IN store_st_id_table THEN ‘Store Staff’
             WHEN person_id IN ticket_st_id_table THEN ‘Ticket Staff’
        END as duty
FROM person
```

Changing Password

```
SET person
UPDATE password = @new_password
WHERE password = @old_password AND person_id = @person_id
```

Deleting The Existing Phone

```
DELETE FROM person_phone
WHERE person_id = @person_id AND phone = @phone
```

Adding a New Phone

```
INSERT INTO person_phone
VALUES(@person_id, @phone)
```

Deleting The Existing Email

```
DELETE FROM person_email
WHERE person_id = @person_id AND email = @email
```

Adding a New Email

```
INSERT INTO person_email
VALUES(@person_id, @email)
```

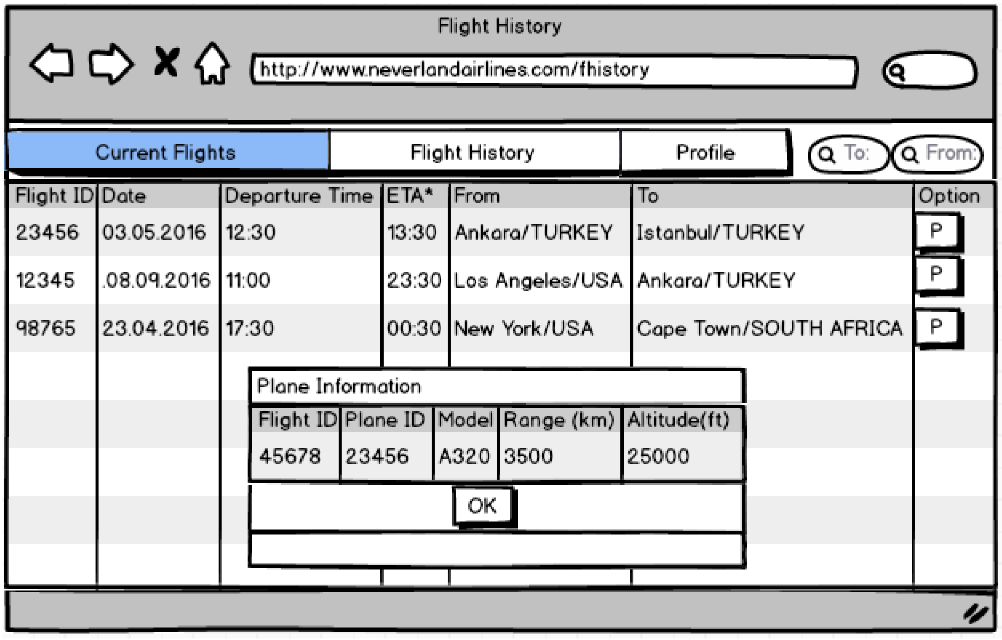
Deleting Account

```
DELETE FROM passenger
WHERE pass_id = @person_id

DELETE FROM pilot
WHERE pilot_id = @person_id

DELETE FROM flight_attendant
WHERE att_id = @person_id

DELETE FROM ticket_staff
WHERE ticket_staff_id = @person_id

DELETE FROM store_staff
WHERE store_staff_id = @person_id

DELETE FROM flight_personnel
WHERE flight_pers_id = @person_id

DELETE FROM staff
WHERE staff_id = @person_id
```

```
DELETE FROM person
WHERE person_id = @person_id
```

Canceling
No query needed.

5.9 Flight Personnel Schedule Screen



Flight ID	Date	Departure Time	ETA*	From	To	Option
23456	03.05.2016	12:30	13:30	Ankara/TURKEY	Istanbul/TURKEY	P
12345	.08.09.2016	11:00	23:30	Los Angeles/USA	Ankara/TURKEY	P
98765	23.04.2016	17:30	00:30	New York/USA	Cape Town/SOUTH AFRICA	P

Plane Information

Flight ID	Plane ID	Model	Range (km)	Altitude(ft)
45678	23456	A320	3500	25000

Figure 5.9 – Flight Personnel Schedule Screen

Inputs: @flight_pers_id, @flight_id

Process: The flight personnel screen has three options. Firstly, flight personnel can see their flight schedules. Secondly, they can see their flight histories. Thirdly, they can see their user profiles, which is the same as the one in the passenger screen. By clicking on the P button, they can view the information about the plane of a particular flight; however this is valid only for pilots. For flight attendants, there is also an option for recording usage of food promotions, but since it is the same with the flight promotion option in the ticket staff screen, it is not shown.

SQL Statements:

Browsing Schedule

```
SELECT *
FROM pilot_schedule
WHERE pilot_id = @flight_pers_id

SELECT *
FROM att_schedule
WHERE att_id = @flight_pers_id
```

In these queries, pilot_schedule and att_schedule views are used which are created in section 6.1.

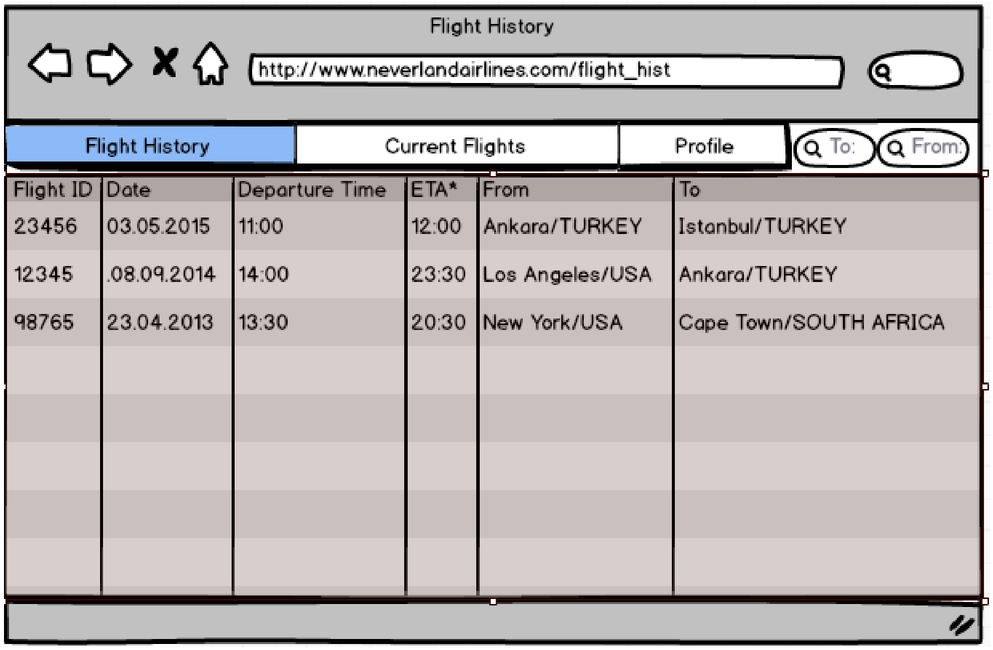
Displaying Information about the Planes of Flights

```
SELECT *
FROM plane_view
```

In this query, plane_view is used which is created in section 6.1.

Canceling
No query needed.

5.10 Flight Personnel Flight History Screen



Flight ID	Date	Departure Time	ETA*	From	To
23456	03.05.2015	11:00	12:00	Ankara/TURKEY	Istanbul/TURKEY
12345	.08.09.2014	14:00	23:30	Los Angeles/USA	Ankara/TURKEY
98765	23.04.2013	13:30	20:30	New York/USA	Cape Town/SOUTH AFRICA

Figure 5.10 – Flight Personnel Flight History Screen

Inputs: @flight_pers_id

Process: In the flight personnel flight history screen, flight personnel can see their previous or current flights.

SQL Statements:

Displaying Flight History

```
SELECT *
FROM pers_history_view
```

In this query, pers_history_view will be used which is created in Section 6.1.

Canceling

No query needed.

In addition to those, flight attendants require the following queries for which user interface is the same with the one belongs to ticket staff.

Displaying Promotions of Passengers

```
SELECT *
FROM food_promotion_view
WHERE pass_id = @pass_id
```

In this query, food_promotion_view is used which is created in Section 6.1.

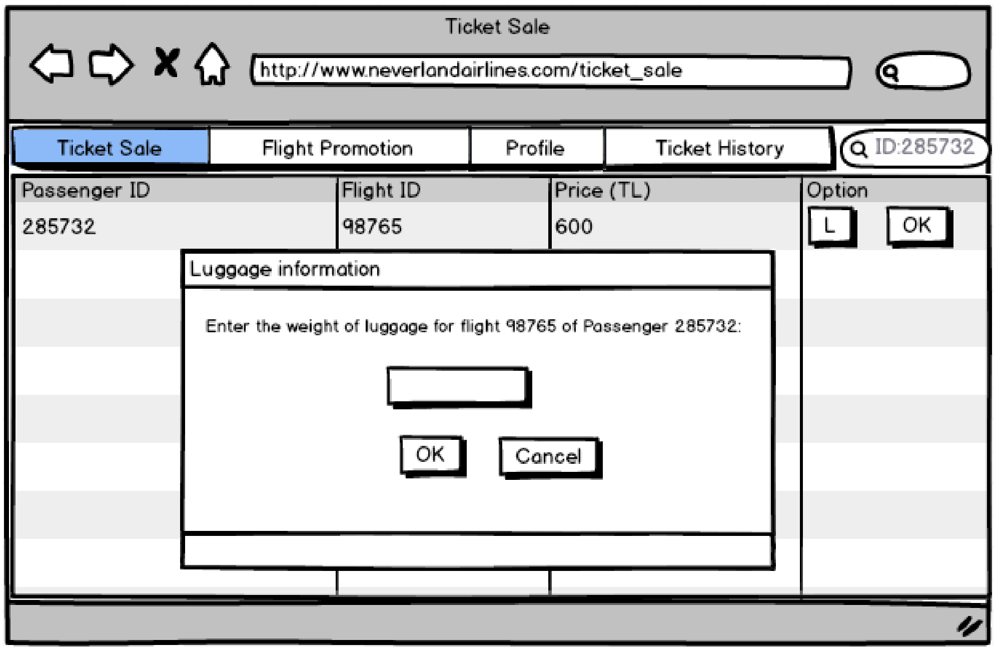
Recording Usage of Food Promotions

```
DELETE FROM food_promotion_view
WHERE prom_id = @prom_id
```

In this query, food_promotion_view will be used which is created in Section 6.1.

Canceling
No query needed.

5.11 Ticket Staff Sale Screen



Passenger ID	Flight ID	Price (TL)	Option
285732	98765	600	L OK

Luggage information

Enter the weight of luggage for flight 98765 of Passenger 285732:

Figure 5.11 – Ticket Staff Sale Screen

Inputs: @pass_id, @ticket_id, @flight_id, @luggage

@pass_id: ID of passenger who uses flight promotion

Process: Ticket staff screen has four options which are recording ticket sales, recording usage of flight promotions, viewing ticket history and viewing user profile. In the sale screen, ticket staff can choose to enter extra luggage weight for a particular passenger by clicking on the L button.

SQL Statements:

Recording Sales

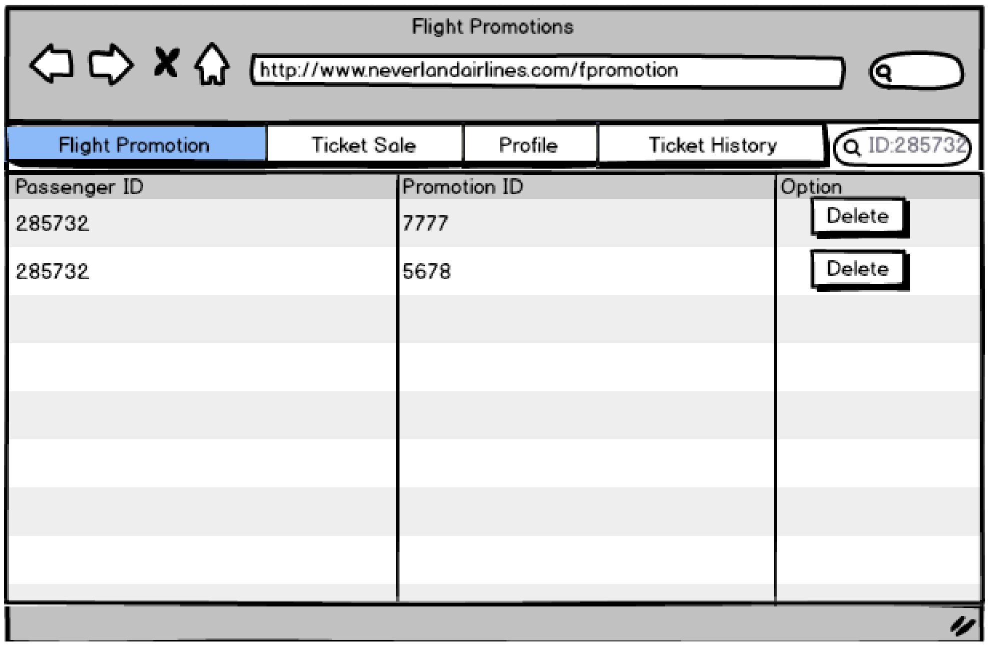
```
INSERT INTO ticket
SELECT flight_id, pass_id, @ticket_id
WHERE flight_id = @flight_id AND pass_id = @pass_id
```

Updating Luggage Weight

```
UPDATE ticket
SET luggage = @luggage
WHERE ticket_id = @ticket_id
```

Canceling
No query needed.

5.12 Ticket Staff Flight Promotion Screen



Passenger ID	Promotion ID	Option
285732	7777	Delete
285732	5678	Delete

Figure 5.12 – Ticket Staff Flight Promotion Screen

Inputs: @pass_id, @prom_id

Process: In the promotion screen, ticket staff will be able to delete the promotions of passengers by searching them with their ID numbers.

SQL Statements:

Displaying Promotions of Passengers

SELECT *
FROM flight_promotion
WHERE pass_id = @pass_id

Recording Usage of Flight Promotions

DELETE FROM flight promotion
WHERE prom_id = @prom_id

For store staff, there are two options which are recording usage of store promotions and viewing user profile, which are the same as the previous ones, so it is not shown.

Canceling
No query needed.

5.13 Admin Flight Screen

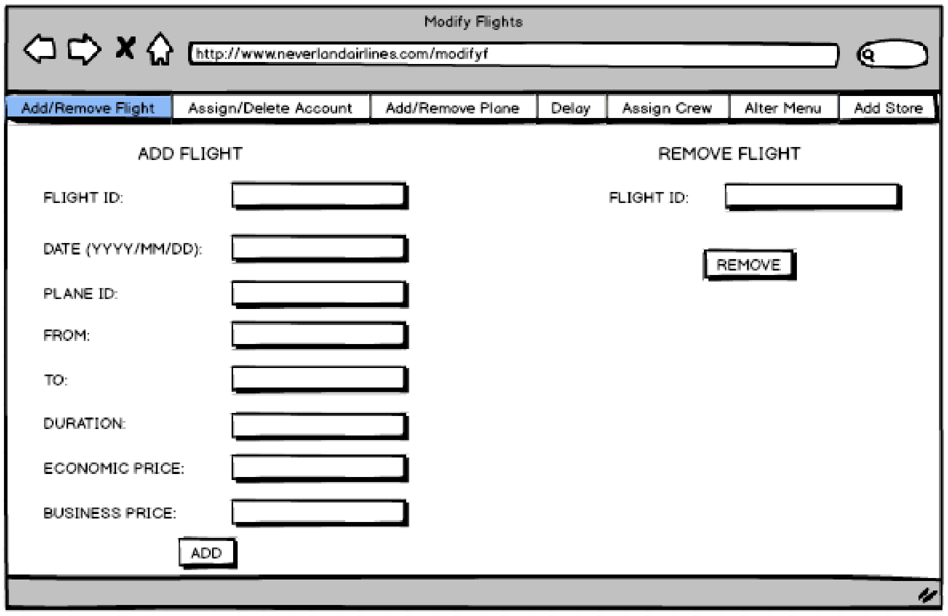


Figure 5.13 – Admin Flight Screen

Inputs: @flight_id, @date, @duration, @econ_price, @business_price

Process: In the admin screen, there are seven options which are adding/removing flights, assigning/deleting accounts, adding/removing planes, delaying flights, assigning crew to flights, altering flight menus, and adding/removing stores. In the flight screen, admin can add flights by specifying their properties, and remove flights by specifying their ID numbers.

SQL Queries

Adding Flights

```
INSERT INTO flight(flight_id, date, duration, econ_price, business_price, dep_airport_id,
arr_airport_id)
VALUES(@flight_id, @date, @duration, @econ_price, @business_price)

INSERT INTO flight_arrival(date, duration, arrival)
VALUES(@date, @duration, @date + @duration)
WHERE NOT EXISTS (SELECT *
                  FROM flight_arrival
                  WHERE date = @date AND duration = @duration)
```

Removing Flights

```
DELETE FROM flight
WHERE flight_id = @flight_id
```

5.14 Admin Plane Screen

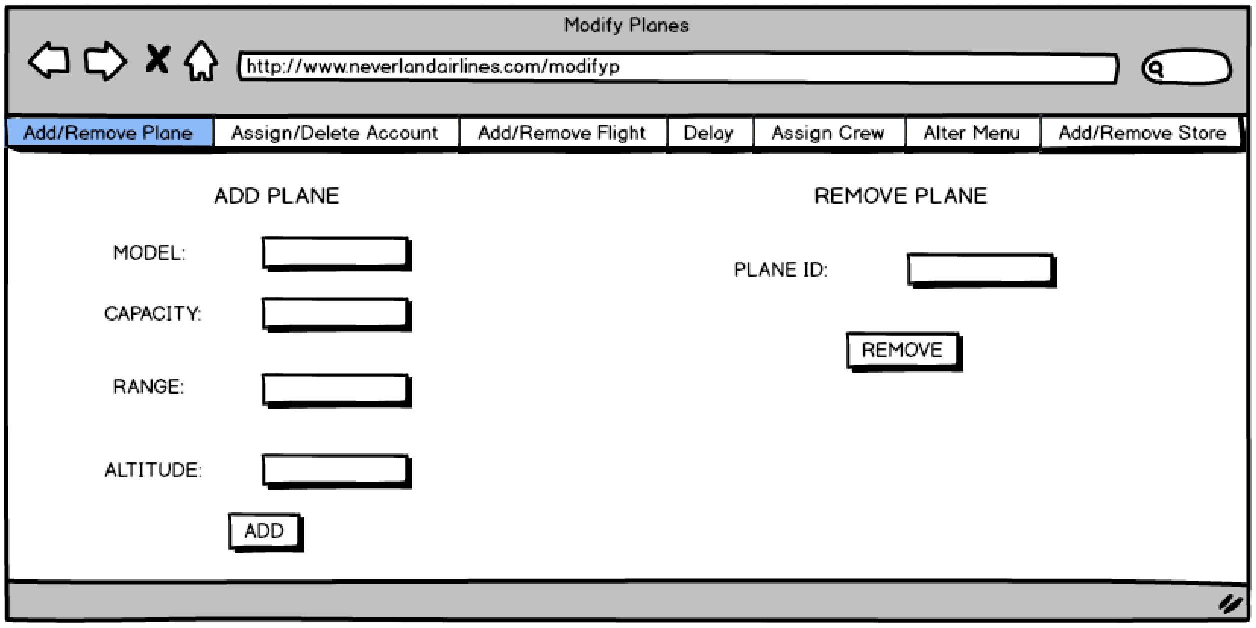


Figure 5.14 – Admin Plane Screen

Inputs: @plane_id, @model, @capacity, @range, @altitude

Process: In the plane screen, admin can add planes by specifying their properties and clicking on the add button. Also, he/she can remove planes by clicking on the remove button.

SQL Queries:

Adding Planes

```
INSERT INTO plane(plane_id, model)
VALUES(@plane_id, @model)

INSERT INTO plane_model(model, capacity, range, altitude)
VALUES(@model, @capacity, @range, @altitude)
WHERE NOT EXISTS (SELECT *
                  FROM plane_model
                  WHERE model = @model)
```

Removing Planes

```
DELETE FROM plane
WHERE plane_id = @plane_id
```

Canceling
No query needed.

5.15 Admin Delay Screen

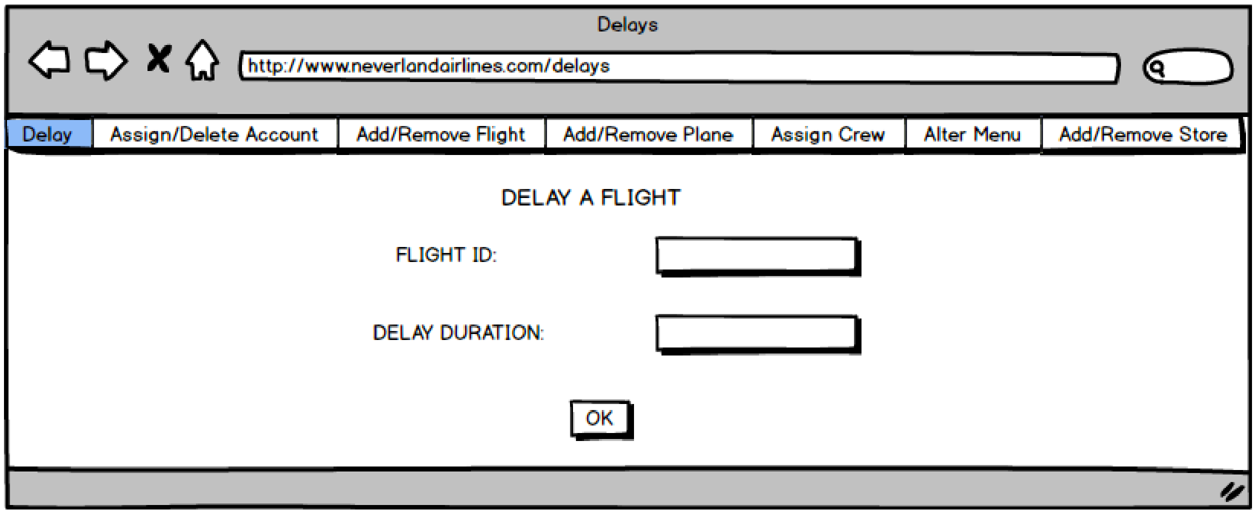


Figure 5.15 – Admin Delay Screen

Inputs: @flight_id, @delay

Process: In the delay screen, admin can delay the flights by specifying their ID numbers and delay durations.

SQL Queries:

Delaying Flights

```
UPDATE flight
SET date = date + @delay
WHERE flight_id = @flight_id
```

A trigger is required to update the arrival and check for the connecting flights.

5.16 Admin Assign Flight Personnel Screen

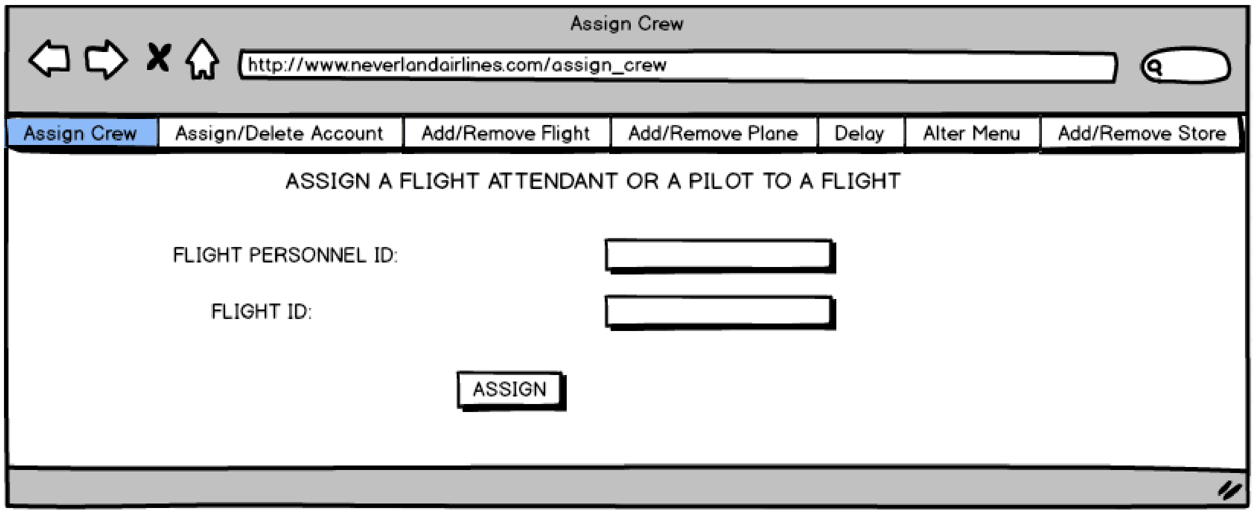


Figure 5.16 – Admin Assign Flight Personnel Screen

Inputs: @flight_personnel_id, @flight_id

Process: In the assign screen, admin can assign flight personnel to flights by specifying their ID numbers and ID numbers of flights.

SQL Statements:

Assigning Flight Personnel to Flights

```
INSERT INTO flight_att
VALUES(@att_id, @flight_id)

INSERT INTO flight_pilot
VALUES(@pilot_id, @flight_id)
```

For the second query, an assertion is required to check if the certificate type and rank of the pilot is convenient for the model of the plane. Also for both of the queries, an assertion is required to check to convenience of cities to the personnel.

5.17 Admin Food Menu Option Screen

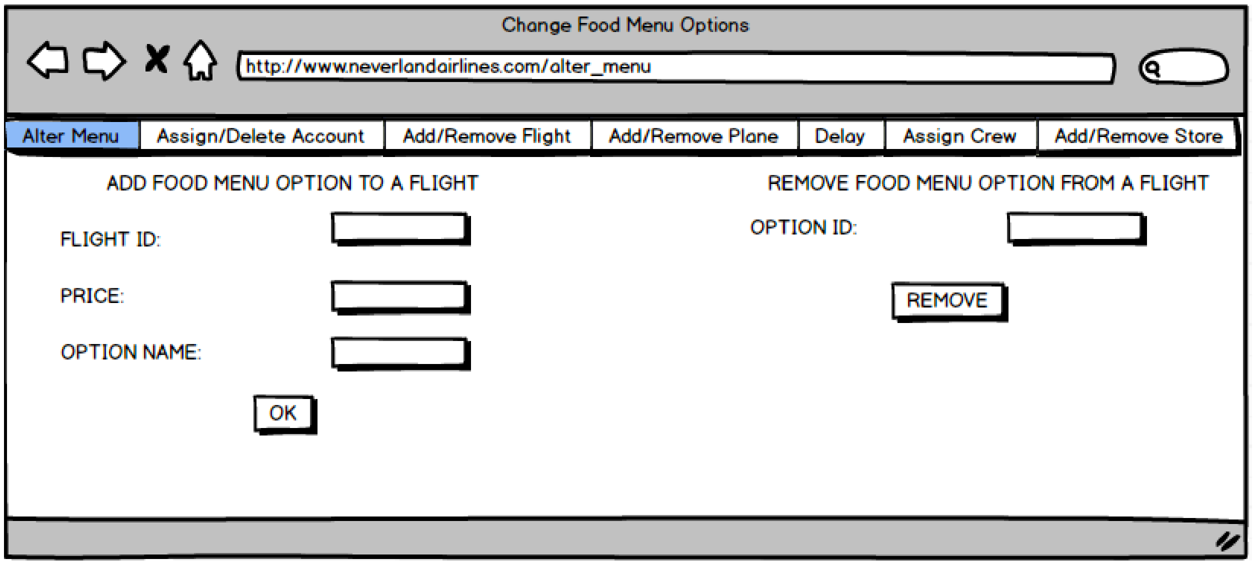


Figure 5.17 – Admin Food Menu Option Screen

Inputs: @flight_id, @option_id, @option_name, @price

Process: In the food menu option screen, admin can add and remove menu options from flights. For adding an option, he/she should specify the flight ID and its price and name. For removing an option, he/she should specify only its ID number.

SQL Statements:

Adding Menu Options

```
INSERT INTO menu_option(flight_id, option_id, option_name, price)
VALUES(@flight_id, @option_id, @option_name, @price)
```

Removing Menu Options

```
DELETE FROM menu_option
WHERE flight_id = @flight_id AND option_id = @option_id
```

5.18 Admin Store Screen

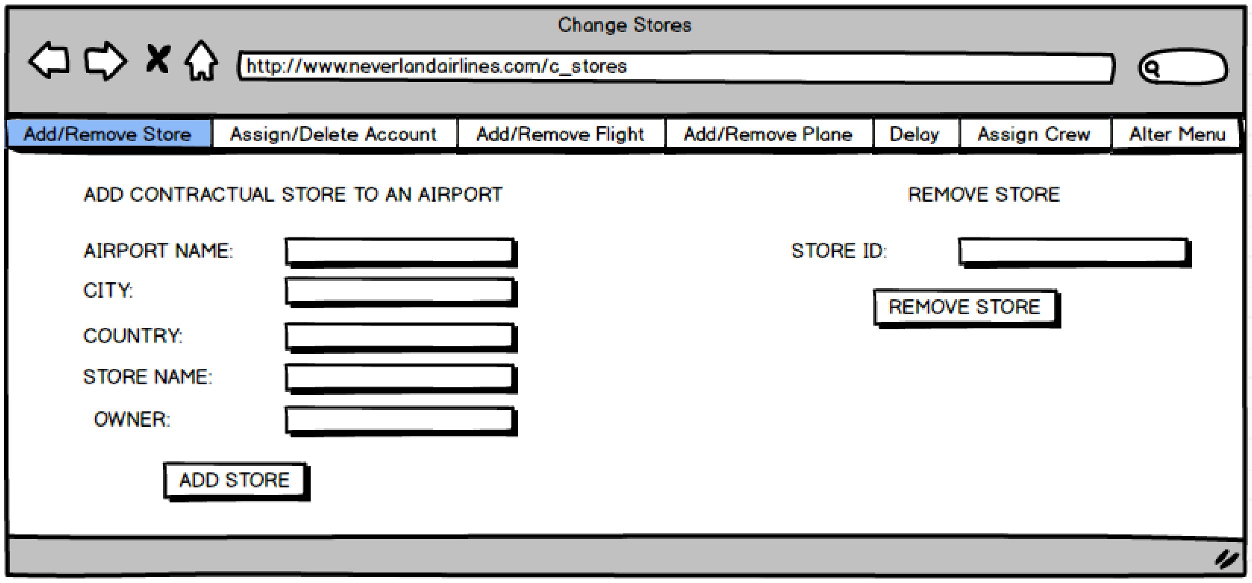


Figure 5.18 – Admin Store Screen

Inputs: @airport_name, @city_name, @country_name, @store_id, @store_name, @owner

Process: In the store screen, admin can add a contractual store to an airport by specifying the airport, and name and owner of the store.

SQL Queries:

Adding Stores

```
INSERT INTO store(store_id, store_name, owner)
VALUES(@store_id, @store_name, @owner)
```

Removing Stores

```
DELETE FROM store
WHERE store_id = @store_id
```

5.19 Admin User Account Screen

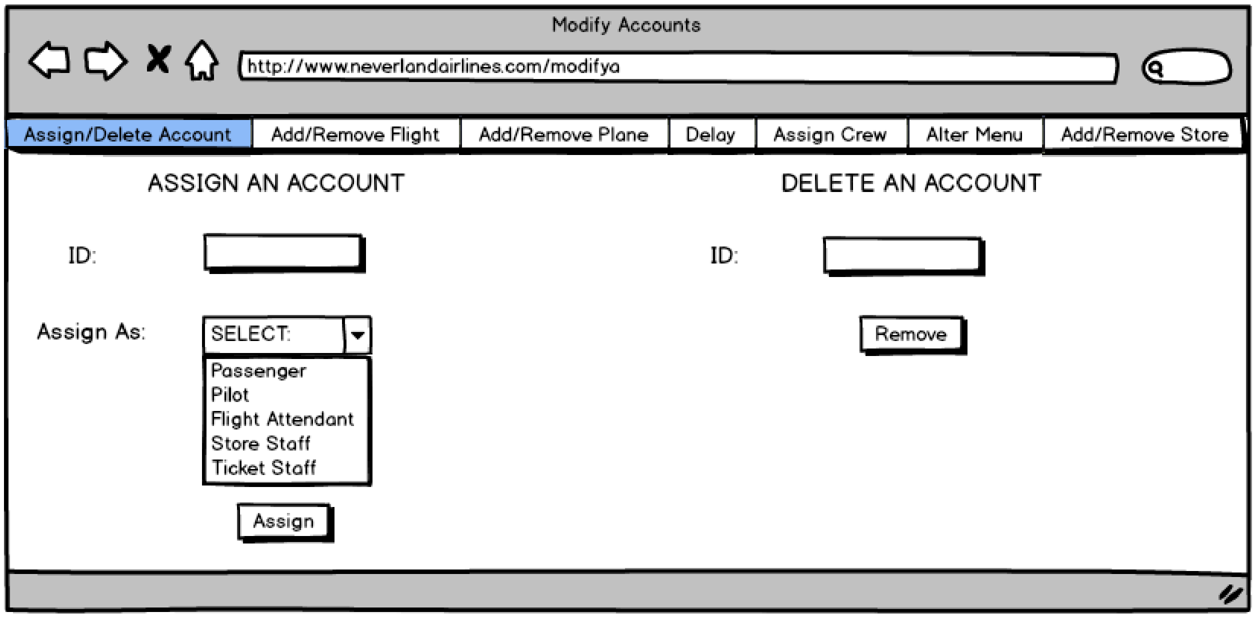


Figure 5.18 – Admin User Account Screen

Inputs: @person_id, @duty

Process: In the user account screen, admin can assign a user to a duty by specifying his/her ID and selecting a duty from the list. Also, admin can delete a user account by specifying his/her ID number. Assigning a person as a passenger is for ending a duty of a company staff, because users can take their accounts as passenger themselves. On the other hand, deleting an account is for security purposes.

SQL Queries:

Removing a Person

It is the same as the query in delete account part in Section 5.3.

Assigning a Person as a Passenger

At first, the account of the person will be deleted by the previous query.

```
INSERT INTO passenger(pass_id, expenditure, prom_expenditure)
VALUES(@pass_id, 0, 0)
```

Assigning a Person as a Flight Attendant

```
DELETE FROM passenger
WHERE pass_id = @pass_id
```

```
INSERT INTO flight_attendant
VALUES(@pass_id, @salary, @start_date, @duty)
```

Assigning a Person as a Pilot

```
DELETE FROM passenger
WHERE pass_id = @pass_id

INSERT INTO pilot
VALUES(@pass_id, @salary, @start_date, @rank, @certificate_type)
```

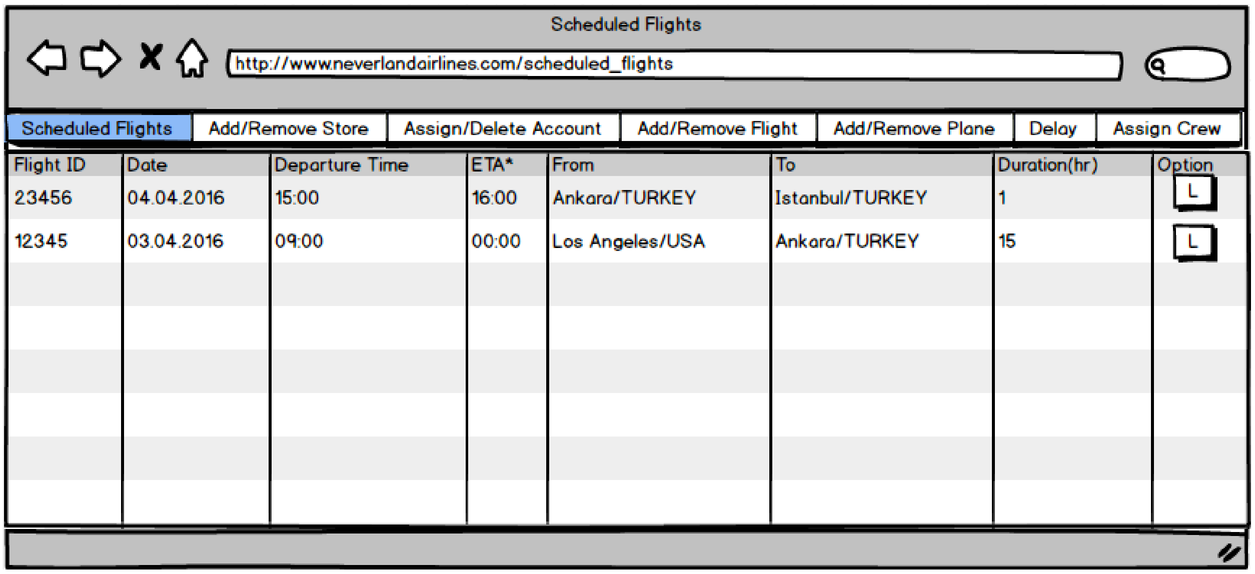
Assigning a Person as a Ticket Staff

```
DELETE FROM passenger
WHERE pass_id = @pass_id

INSERT INTO ticket_staff
VALUES(@pass_id, @salary, @start_date, @ticket_count)
```

Assigning a Person as a Pilot

```
DELETE FROM passenger
WHERE pass_id = @pass_id

INSERT INTO store_staff
VALUES(@pass_id, @salary, @start_date, @sale_count)
```

5.20 Admin Flight Landing Screen



Flight ID	Date	Departure Time	ETA*	From	To	Duration(hr)	Option
23456	04.04.2016	15:00	16:00	Ankara/TURKEY	Istanbul/TURKEY	1	L
12345	03.04.2016	09:00	00:00	Los Angeles/USA	Ankara/TURKEY	15	L

Figure 5.19 – Admin Flight Landing Screen

Inputs: @flight_id, @current_date_time

Process: In the flight landing screen, the current and future flights will be displayed, and admin can set them as landed by clicking on the L button.

SQL Queries:

Displaying Flights

```
SELECT flight_id, date, arrival, duration, arr_city_name, arr_country_name
FROM flight NATURAL JOIN flight_arrival
WHERE landed = false
ORDER BY F.flight_id
```

Landing Flights

```
UPDATE flight
SET landed = true
WHERE flight_id = @flight_id
```

6 Advanced Database Components

6.1 Views

6.1.1 Flight Personnel's Schedule View

Flight personnel should only see the information about the flights that they have scheduled. So, the following view will be useful for this purpose.

```
create view pilot_schedule as
        select *
        from flight
        where flight_id in (select flight_id
                            from flight_pilot
                            where pilot_id = @pilot_id)

create view att_schedule as
        select *
        from flight
        where flight_id in (select flight_id
                             from flight_att
                             where att_id = @att_id)
```

Note: @pilot_id and @att_id are the ID numbers of flight personnel signed in.

6.1.2 Flight Attendants' Promotion View

Flight attendants should only have access to promotions of the passengers who are attending to their flights.

```
create view food_promotion_view as
        select *
        from food_promotion
        where pass_id in (select pass_id
                          from ticket
                          where flight_id in (select flight_id
                                              from flight_att
                                              where att_id = @att_id))
```

Note: @att_id is the ID number of flight attendant signed in.

6.1.3 Pilots' Plane View

Pilots should only have access to the information of the planes which are assigned to their flights.

```
create view plane_view as
        select *
        from plane
        where plane_id in (select plane_id
                           from flight
                           where flight_id in (select flight_id
                                               from flight_pilot
                                               where pilot_id = @pilot_id))
```

Note: @pilot_id is the ID number of the pilot signed in.

6.1.4 Store Staff's Promotion View

If a passenger does not have any reservation for a domestic flight, the ticket staff should not have access to the flight promotions which support only domestic flights such that it is forbidden to use them in the flights which are not domestic.

```
create view flight_promotion_view as
        (select *
        from flight_promotion)
        except
        (select *
        from flight_promotion
        where pass_id not in (select pass_id
                              from reservation natural join flight F
                              where (select city
                                     from airport
                                     where airport_id = F.dep_airport_id) =
                                    (select city
                                     from airport
                                     where airport_id = F.arr_airport_id))
                and domestic = true)
```

6.1.5 History View

Passengers and flight personnel should only have access to the flights in the history table which are related to them.

```
create view pass_history_view as
        select *
        from pass_history
        where pass_id = @pass_id
```

Note: @pass_id is the ID number of the passenger signed in.

```
create view pers_history_view as
        select *
        from pers_history
        where flight_pers_id = @flight_pers_id
```

Note: @flight_pers_id is the ID number of the flight personnel signed in.

6.1.6 Professional Pilot View

Only professional pilots should be assigned to the flights which have planes with some particular models, which means rank of the pilots should be more 10 and their certificate type should be ‘airline transport’. Therefore, a view can be used for abstracting professional pilots.

```
create view professional_pilot as
        select *
        from pilot
        where rank > 10 and certificate_type = ‘airline transport’
```

6.1.7 Connecting Flight View

When a flight is delayed, the connecting flights which are needed to be changed should be determined. For determining connecting flights, it is useful to abstract the ones which have conflicts between the arrival and start times of their flights. The conflicts are determined as if there is not 15 minute time interval between arrival of first flight and start of second flight, then there is a conflict.

```
create view conflicting_connecting_flight as
        select *
        from (reservation natural join flight natural join flight_arrival) R,
              (reservation natural join flight natural join flight_arrival) S,
        where R.pass_id = S.pass_id and R.date < S.date and R.arrival > S.date – 15
```

6.2 Stored Procedures

When passengers’ luggage exceeds a threshold, they need to pay some fee proportional to the amount of luggage. A stored procedure is used for calculating the amount of fee passenger needs to pay. In the procedure, the luggage attribute of the corresponding ticket will be used to calculate the fee.

To determine whether a ticket is domestic or not, a stored procedure will be used. It is necessary for giving flight promotions. In the procedure, the countries of the airports of the flight corresponding to ticket will be compared which each other.

Another procedure will be used to determine the distance of a flight. For this purpose, the distance between latitudes and longitudes of its airports’ cities will be used.

6.3 Reports

6.3.1 Total Number of Ticket Sales and Total Expenditure of Passengers

```
WITH ticket_price(pass_id, pass_name, ticket_id, price)
AS (SELECT pass_id, pass_name, ticket_id,
                CASE
                    WHEN class = ‘econ’ THEN econ_price
                    ELSE business_price
                END AS price
      FROM ticket NATURAL JOIN reservation NATURAL JOIN seat)
SELECT pass_id, pass_name, count(*) AS ticket_count, sum(price) AS total_price,
FROM ticket_price NATURAL JOIN passenger
GROUP BY pass_id
```

6.3.2 Total Number of Flight Personnel for Each Flight

```
WITH pilot_count(flight_id, no_of_pilots) AS
(SELECT flight_id, COUNT(*) AS no_of_pilots
FROM flight_pilot
GROUP BY flight_id),
att_count(flight_id, no_of_att) AS
(SELECT flight_id, COUNT(*) AS no_of_att
FROM flight_att
GROUP BY flight_id)
SELECT flight_id, no_of_pilots, no_of_att
FROM flight_pilot NATURAL JOIN flight_att
```

6.4 Triggers

- When a passenger purchases a ticket, his/her expenditure will be automatically updated. So, when there is an insert in the ticket table, the expenditure attribute of the corresponding passenger tuple will be updated by the price of the flight.
- When a passenger purchases a ticket, a trigger will check if a promotion is available or not and if available, it adds the promotion to the system. So, when there is an insert in the ticket table, the prom_expenditure attribute of the corresponding passenger tuple will be checked to add a tuple to promotion table if necessary.
- When a flight is delayed, its arrival date will be updated. So, when a flight's duration attribute is changed, the corresponding arrival attribute in the flight_arrival table will be updated.
- When a passenger purchases extra luggage, his/her expenditure will automatically be updated. So, when the luggage attribute of a ticket exceeds a threshold, expenditure attribute of the passenger will be updated.
- When a passenger purchases a ticket without reservation, a reservation will automatically be created. So, when there is an insert on the ticket table, there will be an insert on the reservation table if necessary.
- When a flight is landed, it will automatically be removed from the schedules of flight personnel. So, when the landed attribute of a flight is set to true, it will be removed from the flight_pilot and flight_att tables.
- When a passenger purchases a ticket, it will automatically be added to his/her flight history. So, when there is an insert on the ticket table, there will be an insert on the pass_history table.
- When flight personnel are assigned to a flight, it will automatically be added to his/her flight history. So, when there is an insert on the flight_pilot and flight_att tables, there will be an insert on the pers_history table.

An example code for the triggers is shown below. It is the code for update of passenger's expenditure after purchasing a ticket.

```
create trigger expenditure_update
    after insert on ticket
    referencing new row as new_row
    for each row
    begin
            WITH flight_price(econ, business) AS
            (SELECT econ_price, business_price
            FROM flight
            WHERE flight_id = @flight_id)
            ticket_class(class) AS
            (SELECT class
            FROM ticket NATURAL JOIN reservation NATURAL JOIN seat
            WHERE flight_id = @flight_id)
            UPDATE passenger
            SET expenditure = expenditure + (SELECT CASE
                                                WHEN ticket_class.class = ‘econ’ THEN
                                                flight_price.econ
                                                ELSE flight_price.business
                                              END
                                              FROM flight_price, ticket_class)
            WHERE pass_id = @pass_id
    end
```

6.5 Constraints

- Only professional pilots should be assigned to the flights which have planes with some particular models, which means rank of the pilots should be more 10 and their certificate type should be ‘airline transport’.
- In a connecting flight, two flights cannot conflict such that there does not exist any flight whose date attribute is smaller than another, but arrival attribute is larger than the other’s date.
- In the schedules of flight personnel, two flights cannot conflict with each other. That means their dates cannot overlap with each other as in the previous constraint and their cities must be convenient for flight personnel. It means that the city_name and country attributes of the arr_airport of the previous flight should be the same as the city_name and country attributes of the dep_airport of the next flight.

- The distance between departure and arrival places of a flight cannot be more than its planes range. So, the distance between latitudes and longitudes of the cities of airports of a flight cannot be more than the range attribute of the plane of the flight.
- The start dates of flights cannot be earlier than the current date.
- The departure and arrival airports of flights must be different such that their (airport_name, city_name, country) tuples must be different.
- Each flight must have two pilots, which means there must be exactly two tuples with the same flight_id in the flight_pilot table.
- Each flight must have three to five flight attendants, which means there must be three to five tuples with the same flight_id in the flight_att table.
- The number of seats in a flight cannot be more than its plane’s capacity. So, the number of seats with a flight_id cannot be more than the capacity attribute of the plane of the flight with that flight_id.
- The latitude attribute of a city must be between 0 and 360, and longitude attribute must be between 0 and 180.
- The flights whose arrival time is at least 20 minutes later than the current time cannot be marked as landed which means their landed attributes cannot be set to true if the difference between their arrival attributes and the current time is bigger than 20 minutes.

An example code for constraints is shown below which is the code of the professional pilot constraint.

```
create assertion professional_pilot_constraint
check( not exists( select *
                   from pilot natural join flight_pilot natural join flight natural join plane
                   where pilot_id not in (select pilot_id
                                          from professional_pilot)
     and (model = ‘A320’ or model = ‘A380’)))
```

In this assertion, professional_pilot view is used which is created in Section 6.1.

7 Implementation Plan

The following languages will be used in the system:

- Persistent data: MySQL
- Application logic: Phyton
- Application server: Flask
- User interface: React

React is a JavaScript library which is used for providing a view for data and Flask is web framework of Phyton.